DOCUMENT RESUME

ED 397 026 SP 036 742

AUTHOR	Merrill, Douglas C.; And Others
TITLE	Tutoring: Guided Learning by Doing. RAND Reprints.
INSTITUTION	Rand Corp., Santa Monica, CA. Inst. on Education and Training.
REPORT NO	RAND-RP-329
PUB DATE	95
NOTE	63p.; Reprinted from "Cognition and Instruction" (v13 n3 1995).
AVAILABLE FROM	Distribution Services, RAND, 1700 Main Street, P.O. Box 2138, Santa Monica, CA 90407-2138.
PUB TYPE	Reports - Research/Technical (143)
EDRS PRICE	MF01/PC03 Plus Postage.
DESCRIPTORS	College Students; Critical Thinking; Discourse Analysis; *Feedback; Higher Education; *Individual Instruction; Learning Activities; *Learning Processes; *Problem Solving; Programming Languages; Teaching Methods; *Tutoring; Tutors
IDENTIFIERS	LISP Programing Language

ABSTRACT

Individualized instruction significantly improves students' pedagogical and motivational outcomes. The study described here sought to characterize tutorial behaviors that could lead to these benefits and to consider why these behaviors should be pedagogically useful. The experiment studied 16 undergraduate and graduate university students learning LISP programming with the assistance of a tutor. Tutoring sessions were audiotaped, allowing analysis of every verbal utterance during the sessions and identification of the conversational events that led to pedagogical success. This discourse analysis suggested that tutors were successful because they took a very active role in leading the problem solving by offering confirmatory feedback and additional guidance while students were on profitable paths and error feedback after mistakes. However, tutors carefully structured their feedback to allow students to perform as much of the work as possible while ensuring that problem solving stayed on track. These results suggested the types of strategies tutors employ to facilitate guided learning by doing. Instructions for coders and reliability coders and a table showing transitions between events are appended. (Contains 72 references.) (Author/ND)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED 397 026

RAND

REPRINTS

Tutoring

Guided Learning by Doing

Douglas C. Merrill, Brian J. Reiser,
Shannon K. Merrill, and Shari Landes

Reprinted from *Cognition and Instruction*

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☐ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

PERMISSION TO REPRODUCE AND DISSEMINATE THIS MATERIAL HAS BEEN GRANTED BY

E. D. Hill

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

Institute on Education and Training

BEST COPY AVAILABLE

S1036742

The RAND reprint series contains, with permission of the publisher, reprints of research originally published by RAND staff in journals or books. RAND is a nonprofit institution that helps improve public policy through research and analysis. RAND's publications do not necessarily reflect the opinions or policies of its research sponsors. For more information or to order RAND documents, see RAND's URL (http://www.rand.org) or contact Distribution Services, RAND, 1700 Main Street, P.O. Box 2138, Santa Monica, CA 90407-2138, phone (310) 451-7002; fax (310) 451-6915; Internet order@rand.org.

Published 1995 by RAND
1700 Main Street, P.O. Box 2138, Santa Monica, CA 90407-2138

RAND

REPRINTS

RAND/RP-329

Tutoring

Guided Learning by Doing

Douglas C. Merrill, Brian J. Reiser,
Shannon K. Merrill, and Shari Landes

Reprinted from *Cognition and Instruction*

Institute on Education and Training

Tutoring: Guided Learning by Doing

Douglas C. Merrill
Institute for Education and Training
RAND

Brian J. Reiser and Shannon K. Merrill
School of Education and Social Policy
and The Institute for the Learning Sciences
Northwestern University

Shari Landes
Department of Psychology
Princeton University

Individualized instruction significantly improves students' pedagogical and motivational outcomes. In this article, we seek to characterize tutorial behaviors that could lead to these benefits and to consider why these actions should be pedagogically useful. This experiment examined university students learning LISP programming with the assistance of a tutor. Tutoring sessions were audiotaped, allowing us to analyze every verbal utterance during the sessions and thereby to identify the conversational events that lead to pedagogical success. This discourse analysis suggests that tutors are successful because they take a very active role in leading the problem solving by offering confirmatory feedback and additional guidance while students are on profitable paths and error feedback after mistakes. However, tutors carefully structure their feedback to allow students to perform as much of the work as possible while the tutor ensures that problem solving stays on track. These results suggest the types of strategies tutors employ to facilitate guided learning by doing.

Novices often have great difficulty mastering new domains. It is generally accepted that the best way to acquire new domain skills is by solving problems (Anderson, 1983; Laird, Rosenbloom, & Newell, 1986; VanLehn, 1988), but there are dangers inherent in this sort of learning by doing. Floundering during

Requests for reprints should be sent to Douglas C. Merrill, RAND, 1700 Main Street, Santa Monica, CA 90407-2138. e-mail: Doug_Merrill@rand.org

Reprinted by permission from *Cognition and Instruction*, Vol. 13, No. 3, 1995, pp. 315–372. Copyright © 1995 Lawrence Erlbaum Associates, Inc.

problem solving often leads to working memory overload, which interferes with learning (Sweller, 1988). Furthermore, errors during problem solving can often engender confusion and frustration. It is very difficult to learn from problem-solving episodes that consist largely of attempts to recover from errors (Anderson, 1983; Lewis & Anderson, 1985). Ameliorating these costs would allow students to gain the maximum benefits of learning by doing.

Individualized instruction, or tutoring, considered by many to be the best method of instruction, is one method for minimizing these costs (B. S. Bloom, 1984; P. A. Cohen, Kulik, & Kulik, 1982; Lepper, Aspinwall, Mumme, & Chabay, 1990). Individualized instruction has both motivational and cognitive benefits. For example, tutoring leads students to feel more competent (Lepper & Chabay, 1988). This feeling appears to be justified; tutored students perform 2 standard deviations higher than their colleagues receiving traditional instruction, indicating that almost all tutored students perform better than the mean performance of students receiving classroom instruction (B. S. Bloom, 1984). Yet although these pedagogical benefits have been noted, their source has not been adequately characterized. Our present goal is to investigate the strategies tutors use that can lead to these benefits.

Merrill, Reiser, Ranney, and Trafton (1992) argued that tutors balance the goal of allowing students to perform as much of the problem solving as possible with the goal of ensuring that problem solving remains productive. Rather than letting students solve problems on their own, with occasional advice, tutors carefully monitor the problem solving to ensure that it stays on track and to help direct students back toward a productive solution path when needed. Thus, tutors offer a kind of *guided learning by doing* that enables their students to attain the benefits of learning by doing while avoiding some of the costs. We examine here how tutors achieve these benefits through careful guidance and characterize the way tutorial interactions lead to strong learning advantages for students.

We describe the ways in which tutors guide and assist students' learning while they solve problems and work to understand new material. Furthermore, we characterize the situations in which this guidance takes place. In fact, this guidance is not always easy to identify. Examining an actual tutoring session reveals the complexity and subtlety of the tutor's role. Example 1, a transcript of a student–tutor discussion of LISP, shows a very interactive relationship between student and tutor, with the student and tutor interrupting each other frequently and occasionally completing each other's sentences.

Example 1 (transcript of a LISP tutoring session):

1. *Student:* [typing]
   (defun classify-sentence (sent)
2. *Tutor:* So, that's a very long function name!
3. *Student:* [typing]
   (cond ((or (equal (car sent) 'why)
   (equal (car sent) 'how)) 'question)
   ((and (member 'was sent)

(member 'by sent)) 'passive)
(t 'active)))
All-righty. Yeah. [typing] *(classify sentence (mary*

4. *Tutor:* You have to quote the thing here, or else it'll think it's a function call. [pause] They give you some examples, if you wanna use theirs,=
5. *Student:* =Oh, sure.= [typing]
<DEL><DEL><DEL><DEL><DEL>
'(mary threw the snowball at steve))
6. *Tutor:* = or you could just make up some.
7. [Computer returns "Undefined variable sentence"]
8. *Student:* Oops. [pause] Oh, I forgot to put the dash.
9. *Tutor:* Yup! So it thought it was a variable.
10. *Student:* [typing] *(classify-sentence '(mary threw the snowball at steve))*
Nice!
11. *Tutor:* Good.
12. *Student:* I think I'll try one more now.
13. *Tutor:* Do you understand the difference between *and* and *or* now?
14. *Student:* Uhh . . . let's see.
15. *Tutor:* [unintell] //
16. *Student:* If . . *and* needs both of them to be true, and then it returns true.
17. *Tutor:* Um hmm.
18. *Student:* And *or* just needs one of them to be true, and it returns true.
19. *Tutor:* Right.
20. *Student:* But if both of them are *nil* in *or*, then it would return *nil*.
21. *Tutor:* Right. And in both of them, *or* and *and*, it doesn't necessarily return the letter *t*. It'll return whatever true value that it gets to.
22. *Student:* Uhh . . . I wonder—I wonder how that worked in my function that I just wrote.
23. *Tutor:* That's fine, because the *cond* knows, *cond* knows that anything that's not *nil* is like true.

We examine tutorial guidance in a number of key sites for learning. First, we show how tutors respond when students are on an appropriate solution path. Perhaps tutors concentrate their assistance on helping students to realize what they have done correctly and to understand its consequences. For example, the tutor commented that a step was correct in Utterances 11, 17, and 19 of Example 1, thus encouraging the student to continue with that path in the problem solving, and elaborated on a correct student action in Utterance 9.

Then we turn to an examination of student–tutor interactions that follow a problem-solving difficulty. Do tutors provide feedback after impasses? If so, what form does the tutorial guidance take: Are tutors very directive or subtle? Do they allow students to find and repair their own mistakes, offering error feedback only when the student asks, or do they instead intervene frequently to point out errors? In Utterance 4 of Example 1, for instance, the tutor noticed that the student had failed to include a required quotation mark and simply told the student how to fix it, thus not allowing the student to find the error herself.

The goal of this investigation is not only to characterize the range of tutorial strategies but also to identify the factors influencing when tutors intervene and the intervention strategies they use. It should be possible to determine these factors by identifying patterns and consequences of tutorial intervention. Several researchers have identified tutorial guidance methods (e.g., Fox, 1991; Graesser, Person, & Huber, 1993; Lepper et al., 1990; Lepper & Chabay, 1988; Littman, Pinto, & Soloway, 1990; McArthur, Stasz, & Zmuidzinas, 1990). By and large, these researchers have focused on a few central episodes that occurred during longer tutoring sessions. Each of these analyses has highlighted some of the characteristics of tutorial behavior, thus revealing particular aspects of tutorial strategies rather than a broad range of behaviors and the situations in which they arise. Our work examines behaviors in a larger context over a longer period of time to capture the interplay of these and other tutorial behaviors with student problem solving. We first review the various views of tutoring suggested by previous research, displaying the complexity of factors affecting tutorial behavior, and then formulate the questions that drive our approach.

Fox (1991) argued that tutors provide a "safety net" for students, keeping them from going off track by offering frequent confirmatory feedback. Tutors provided a confirmation (e.g., "Yes") to each student step, but if it was delayed by as little as a second after the step, the student presumed an error had occurred and began a repair. The tutor helped with the repair as needed, even to the extent of providing the correct answer if necessary. Generally, however, tutors tried to remain as subtle and unobtrusive as possible. Thus, Fox characterized feedback as primarily (though not completely) confirmatory, keeping the student going on productive paths. The absence of confirmatory feedback was seen by students as a signal that an error had occurred.

Lepper and his colleagues also considered how tutors scaffold students' learning, but they concentrated on the motivational aspects of tutorial feedback (Lepper et al., 1990; Lepper & Chabay, 1988). They argued that a major goal of tutors is to keep students from becoming discouraged and from blaming themselves when problem-solving difficulties are encountered. The tutors accomplished this in two ways. First, they emphasized that the problems were hard, thereby redirecting the blame from the students to the problems and permitting students to attribute the errors to the difficulty of the problems rather than to a lack of ability. Second, rather than telling students how to repair errors, Lepper's tutors asked leading questions that helped students identify and repair errors themselves. Similarly, some teachers use questions and counterexamples to help students uncover faults in their own reasoning (Collins & Stevens, 1982; Collins, Warnock, & Passafiume, 1975). These analyses suggest that tutors keep students feeling successful by allowing them to find and repair errors, thereby maintaining their sense of control over the problem solving (cf. Scardamalia, Bereiter, McLean, Swallow, & Woodruff, 1989), and to blame errors on external factors. The Fox (1991) and Lepper et al. (1990) studies suggest that much of the work in tutoring sessions is performed by the

student, even after errors. The tutor takes advantage of opportunities to help students remain sure of themselves and their problem-solving success and to ensure that students notice any errors. The students are primarily responsible for repairing errors, but the tutor will scaffold the process as needed by asking leading questions and providing the occasional correct answer.

Putnam (1987) also argued that tutors are primarily interested in getting students to complete the material. Putnam proposed, however, that tutors do not rely chiefly on opportunistic planning but rather on *curriculum scripts* that suggest a loosely ordered set of tasks to perform during a session to guide the session. These curriculum scripts are plans that vary little across sessions. In this view, errors are not particularly important opportunities to increase student understanding. Instead, tutors try to get students back on to a correct track by giving the answer to the problem.

Much as Fox (1991) and Lepper et al. (1990) focused on the active role of the students, Graesser et al. (1993) argued that tutoring is successful because sessions are structured to allow students the opportunity to learn actively through their own questions. Indeed, students ask approximately 100 times more questions during tutoring than in classroom situations (Kerry, 1987), and learning through asking questions may be superior to more passive learning (Graesser et al., 1993). Interestingly, students often fail to understand questions they are asked or the answers to their own questions, and thus tutors must collaborate with students to clarify the meaning of questions and answers. Graesser (1993) argued that these interactions of question, answer, and collaborative search for meaning form a five-step script, called a *dialog frame*. Dialog frames are employed throughout tutoring sessions to guide students' knowledge acquisition and problem solving.

Other researchers have emphasized the role of errors in triggering curriculum scripts. For example, Littman et al. (1990) provided tutors with students' PASCAL programs and asked them to plan an intervention. They found that the tutors structured the entire interaction around feedback for errors. The tutors used a great deal of domain knowledge about the causes and severity of errors to decide on an order for remediating them and planned to offer very directive feedback during the remediation (Littman, 1991). In fact, these tutors used *tutorial planning schemas* based on these errors that guided the tutorial sessions. Planning schemas arise both from domain knowledge and from tutoring knowledge and capture, for example, the fact that repairing an error might be necessary before some other error could be examined, or the notion that several errors might be indicators of the same deep confusion. These schemas allow tutors to develop an optimal structure for the tutoring session that maximizes the success of student repairs.

McArthur et al. (1990) and Schoenfeld, Gamoran, Kessel, and Leonard (1992) also argued that tutors use scripts to guide their behavior but their analyses extend to other kinds of student behaviors besides errors as triggers for a script. These scripts, sometimes called tutorial *microplans* (McArthur et al., 1990), can be triggered by various actions, including errors, new problem-solving goals, and

BEST COPY AVAILABLE

pedagogical goals. Like tutorial planning schemas, microplans are used to decide how to respond to a student action, with each microplan generating one or many tutorial responses. Because microplans can be activated by actions other than errors, however, they offer tutors the flexibility to respond to students' individual needs and confusions while still accomplishing general pedagogical goals. For example, according to McArthur et al. (1990), tutors often remind students of what they are doing and why it is being done, thereby keeping them aware of problem-solving goals.

This brief review of tutoring has revealed two foci of tutoring research. Some researchers (e.g., Fox, 1991; Lepper & Chabay, 1988) have concentrated on the content of tutorial utterances, whereas others (e.g., Littman et al., 1990; McArthur et al., 1990; Putnam, 1987; Schoenfeld et al., 1992) have chiefly considered the ways that tutors organize sessions. These two dimensions are essentially independent—tutorial scripts do not necessarily specify the type of feedback to be given.

These studies have revealed a range of tutorial behaviors, including a focus on positive outcomes (Fox, 1991), the subtle nature of tutorial guidance (Fox, 1991; Lepper et al., 1990), the assistance of tutorial guidance in helping to structure the problem solving (McArthur et al., 1990; Putnam, 1987; Schoenfeld et al., 1992), and response to student errors (Littman et al., 1990). Many of these findings reveal the active role of students in problem solving (Fox, 1991), questioning (Graesser et al., 1993), and repairing errors (Fox, 1991; Graesser et al., 1993). Part of this variation in tutorial behaviors is due presumably to variations in the problem-solving context. These studies varied in factors such as the ages of students and tutors, whether the session was remedial or was covering the material for the first time, and whether the students' participation was voluntary or required.

In general, these tutoring studies have examined portions of tutoring sessions, characterizing certain interactions of theoretical interest. Although these snapshots of tutoring have cast much light on the tutorial process, the complete picture of tutoring has yet to be developed. Developing a model of tutoring that characterizes the ways in which tutorial assistance leads to pedagogical success requires examining tutoring over a long period of time with a variety of students to examine the various contexts in which different tutorial behaviors may arise. Presumably, all the theories just described capture different aspects of the range of tutorial behavior. Previous work has not yet analyzed complete tutorial sessions with the aim of characterizing the contexts giving rise to the full range of tutorial assistance.

The goal of our study is to describe the situations that lead tutors to behave in the ways we have just reviewed. We argue that tutors guide problem solving principally in two ways, with situational characteristics affecting the way chosen and the formation of the guidance. First, tutors offer rapid and explicit feedback to student actions, telling the student whether or not the action was correct. Second, in the event of a mistake, students and tutors collaborate in repairing the error. Tutors carefully choose feedback to allow students to perform many components of the error recovery process (Merrill et al., 1992). We use our

behavioral findings along with other problem-solving research to offer a potential explanation for the success of tutoring.

To investigate these issues, we designed a controlled learning task in which computer novices learned basic programming concepts with the assistance of a tutor. The data presented here were gathered in an experiment contrasting students working a preset curriculum of LISP programming problems in four different learning environments. In this article, we present data from two of the four conditions. The first condition was a traditional one-on-one tutoring situation, in which students solved LISP programming problems with the constant assistance of a tutor. To allow us to explore tutors' means of guidance, we audiotaped all verbal interactions between student and tutor. We used two different tutors, each with significant tutoring experience, to increase the diversity of behaviors that would be revealed by the discourse analysis of the tutorial interactions. The goal of this analysis is to characterize tutorial actions in long-term interventions that cover a wide range of material. Thus, we chose to emphasize depth of interaction with each tutor rather than number of tutors.

As we have noted, it is plausible that many factors affect tutorial behavior, such as the domain being studied, the age of the students, and whether the session is remedial or not. This study uses nonremedial sessions and college-age students, thus representing one possible subset of the space of tutorial possibilities. We focused on students learning new material because these sessions are likely to encompass a range of problem-solving scenarios, including those in which the student succeeds and those in which the student encounters obstacles.

The control condition for this article, called the independent problem-solving condition, consisted of novices covering the same material and solving the same problems, but without tutorial assistance. Verbalizations were not recorded in this condition.

This study includes approximately 50 hr of student–tutor verbal interactions. To analyze these data, we used a style of discourse analysis similar to protocol analysis (Ericsson & Simon, 1984) to examine the contexts in which different tutorial actions arose and the outcomes of these actions. Our version of this technique is very similar to discourse analysis in that it analyzes the language of two or more people talking during problem solving to reveal the actions employed during these dialogues. The important theoretical claim of both protocol analysis and our approach is that the researcher cannot presume to have full access to the mental states of the problem solvers and so must focus solely on information that is completely explicit in the participants' utterances.

By looking at patterns of utterances, a researcher attains limited access to the mental processes used while solving a problem based only on the information to which the participant is attending, as well as any inferences, assertions, or questions made explicit by the participant. The researcher develops categories based on the explicit content of each utterance—not what the researcher believes the speaker meant—and categorizes all utterances made during the task (Bakeman & Gottman, 1986). These categorizations specify the various problem-solving

events that occur on the way to a solution. This technique allows us to look for contingencies between problem-solving context and tutorial action by examining the transitions from one sort of event to another.

We developed a coding scheme containing 36 categories designed to capture the full range of both student and tutor behaviors during problem solving. For example, we had categories for utterances such as a student asking for help, setting a goal, or generating a concrete example, as well as for tutorial actions such as error feedback or goal setting (the complete scheme is described later). We categorized each utterance made by tutor or student during the approximately 50 hr of sessions. Thus, this study presents a fine-grained picture of tutoring over an extended period of problem solving.

These extended microanalyses enable us to examine tutorial behaviors across many different contexts within each student and with different students to determine which aspects of the behaviors are components of successful tutoring in this type of domain. In sum, this study enables us to characterize the ways tutors assist problem solving in procedural domains and to develop a model to show how these behaviors make tutors so successful.

METHOD

Participants

The participants in this experiment were 16 Princeton University undergraduates and graduate students recruited through sign-up sheets on campus (8 in the one-on-one tutoring condition and 8 in the independent problem-solving condition). Students were randomly assigned to conditions and were paid $5.00 per hour for their participation. The participants included an equal number of men and women, all with no previous programming experience. To minimize individual differences across conditions, participant sex and SAT Math score, a good predictor of success in learning to program (Mayer, Dyck, & Vilberg, 1986), were roughly balanced across conditions. The overall mean Math SAT of the participants was 690.

Students in the one-on-one tutoring condition were matched with a tutor of the same sex. Two Princeton University undergraduates acted as tutors in this experiment. The female tutor had previous experience tutoring math and science in high school and was an experienced LISP programmer. The male tutor had experience teaching LOGO to students in summer camps; he was also an experienced LISP programmer. Both tutors were unaware of the goals of the study.

Materials

The students worked 56 problems interspersed throughout the first three chapters of an introductory LISP textbook, *Essential LISP* (Anderson, Corbett, & Reiser, 1987). The three chapters amounted to roughly 50 pages of text and introduced

25 built-in LISP functions, variables and constants, the form of basic function definitions, and the use of conditionals.

We constructed two cumulative posttests that covered material in the second and third chapters. These pencil-and-paper posttests consisted of problems requiring students to generate LISP programs to solve small problems, to find and repair errors in previously generated programs, and to give the output of LISP functions with given input values.

At all times during the learning session, the students were able to work on a computer terminal running a LISP interpreter that had been modified to store and timestamp all keystrokes the students made. The interpreter did not contain the traditional LISP debugging mode, which often confuses novices. There was also a simple screen editor available for the students to use to edit function definitions.

Procedure

Students were told to read the material in the textbook and to attempt to solve the problem sets intermixed in the chapters. The students received a demonstration of the computer system at the beginning of the first session and a demonstration of the editing facilities at the beginning of the second session. In addition to the 56 assigned problems, all students took the untimed posttests after the second and third chapters. Students were allowed to work at their own pace and took between 5 and 10 hr to complete the task, distributed over 3 to 5 days. They were free to refer to the text at any time. All students completed all problems correctly.

One-on-one tutoring. Participants in the one-on-one tutoring condition worked through the material with the assistance of an experienced human tutor. The tutors were instructed to use the textbook and all 56 assigned problems but were not told to use a particular method of tutoring with the students; instead, they were to rely on their tutoring expertise. The student and tutor were seated side by side at a table containing the computer terminal and a tape recorder. The keyboard was placed in front of the student to facilitate the student's typing, but the tutor could type, if needed. The tutors were instructed to require the students to solve each problem correctly before moving on to the next problem in the preset sequence.

Independent problem solving. In the independent problem-solving condition, students worked through the problems without access to a tutor. The experimenter checked the solutions after each chapter and told the students which problems, if any, were incorrect. The experimenter did not convey anything about the errors in the solution but simply reported that the solution was incorrect. The students were then required to make the necessary repairs. Thus, participants in this condition were also required to solve all 56 assigned problems correctly. The students were allowed to ask the experimenter questions if they felt completely confused. In these infrequent cases, the experimenter would offer some

small amount of assistance, such as pointing the student back to the relevant section of the textbook.

Discourse Analysis Methods

To analyze the discourse between tutors and students, we first transcribed the complete protocols from all 8 student–tutor pairs. Then, we interspersed the records made by the computer of all LISP interactions into the transcriptions to provide one complete trace of all verbal behavior and interactions with the computer. This complete trace serves as the data for this analysis. The goal of this analysis is to uncover the behaviors giving rise to tutorial effectiveness and the situations in which these behaviors occur by examining the patterns of student–tutor utterances throughout the problem solving.

Before discourse analysis was performed, all transcripts of verbalizations were divided into smaller units corresponding to codable events. Then, each segment was categorized according to the type of student or tutorial action. This process is known as *segmentation* (Bakeman & Gottman, 1986). When dividing a protocol into segments, the segmenter must decide when one segment ends and another begins. Explicit segmentation rules are used to ensure reliable segmentation, typically by requiring segmenters to make few inferences (Bakeman & Gottman, 1986). One way to measure the success of these rules is to measure percentage agreement, capturing the extent to which segmenters divide the protocol similarly.

In this study, we used a method for breaking the discourse into events we call *segmentation by idea*, based on identifying when a speaker is discussing a new point. All discourse on any one point by one speaker becomes one segment. Thus, each time a new idea is entered into the discourse, a new segment is created, allowing detailed access to each topic of any speaker's discourse.

Segmentation by idea differs from turn-taking segmentation, a very common scheme (e.g., L. Bloom, Rocissano, & Hood, 1976), in that turn-taking methods typically attempt to take the whole utterance of a speaker (one turn) as the unit of analysis to be categorized, whereas segmentation by idea allows a single utterance to be broken up if it expresses multiple points.

Example 2 (a student–tutor interaction after the segmentation and categorization process):

1. TFA	*Tutor:*	So in this example down here? See how they have two separate=
	Student:	Yeah.
	Tutor:	=parameters.
2. SC	*Student:*	Yeah.
3. Comm	*Tutor:*	. . . So, if you
4. SC	*Student:*	That makes sense.
5. TSP	*Tutor:*	called *insert-second* on, like, . . . *dog*, and then the list *(bird cat egg)*,

	Student:	Hmm.
	Tutor:	=then *item* would always refer to *dog*, for your function call,=
	Student:	Right.
	Tutor:	=and *oldlist* would always refer to that list, bird, cat, whatever.
6. SC	*Student:*	OK.
7. TSG	*Tutor:*	And then you have to figure out what exactly you want it to do.
8. SC	*Student:*	Right.
9. TElab	*Tutor:*	Using those functions we learned yesterday. [pause]
10. TFA	*Tutor:*	This is a, this is a good example, I like this . . . thing. 'Cause this shows how LISP is actually going through and interpreting it.
11. TSP	*Tutor:*	So, let's say you typed in this, umm, function call . . . function definition of *double*. Telling you the parameter is *num*, so there's only one parameter, you're only going to have one argument. But then if they call, if you call *double*, on this other function, it's kind of interesting to see how it actually evaluates that because this whole list, *(+ 5 10)*, is going to eventually be assigned to *num*.
12. SC	*Student:*	OK.
13. TSP	*Tutor:*	Umm, but first, okay, it looks at *double*, it knows this is the definition it's gonna use, and then it has to evaluate that argument. So it works inside-out, like it did, like we were looking at yesterday.
14. TSP	*Tutor:*	It figures out what 5 plus 10 is, gets 15, then it assigns 15 to *num*, binds *num*
	Student:	Mmm hmm.
	Tutor:	to 15, that's the words they use, and uhh, . . . so then, in the rest of the body, [laughs] that one line, *num* substitutes—is substituted with 15. So it looks at the body, *(* num 2)*, *num* is evaluated and *num* gets 15, 2 stays itself, and then it applies the multiplication, multiplies 15 by 2, and this line returns 30. Now whatever the body of the function returns, the whole function will return, so actually 30 will get printed out there.
15. SC	*Student:*	That makes sense.

Consider, for example, the protocol shown in Example 2, which is both segmented and categorized (abbreviations are described later and appear in Appendix B). In this part of the session, the tutor is explaining how LISP matches parameters in a student's function to actual values when that function is called. Note that a single tutorial utterance was divided into three categorizable segments (Events 9 to 11 in Example 2), whereas it would have been classified as a single turn in a turn-taking scheme. Segmentation by idea allows individual problem-solving events that occur in a series within one participant's comments to be considered

BEST COPY AVAILABLE

separately rather than together. Conversely, a segment can continue across an utterance of the other person if the speaker fails to acknowledge the second person's speech in any way. Consider, for example, Event 5 in Example 2. Here the tutor does not verbally acknowledge any of the student's comments and continues describing the same concept. Thus, this entire interaction was segmented as one event. This method thus allows each segment to capture a single complete problem-solving event.

To ensure the reliability of our scheme, two of us (D. Merrill and S. Merrill) independently segmented all the protocols, and differences between segmentations were resolved between us. Instructions for segmentation are shown in Appendix A. We initially agreed on 98% of segments, calculated across all protocols, indicating that the rules we used could be implemented very reliably (Bakeman & Gottman, 1986) and that there were very few differences to resolve. This study analyzed approximately 15,000 segments.

After segmenting all protocols, we assigned each segment to 1 of 36 categories that captured each student or tutor action. We developed these categories to represent the information expressed throughout the problem solving. They enabled us to specify the problem-solving contexts that lead to the various tutorial behaviors and to examine the pedagogical strategies of the tutors. We designed categories to capture tutorial behaviors highlighted as crucial for pedagogical success in the theories of tutoring presented earlier. For example, we created categories for tutor confirmatory feedback (Fox, 1991), tutor motivational feedback (Lepper et al., 1990), and tutor goal remindings (McArthur et al., 1990). We also designed categories for events viewed as important to learning in general, such as making assertions, trying solutions, setting goals, and offering explanations. Definitions and examples of each category are included in Appendix B. Each category was designed so an utterance could be classified by the explicit meaning of the utterance.

The categories of student actions in the student–tutor discourse are:

1. The student constructs a solution to a problem (Student Problem-Solving Action).
   a. Student Correction (SCr).
   b. Student Elaboration (SElab).
   c. Student Example (SE).
   d. Student Focuses Attention (SFA).
   e. Student Indicates Difficulty (SID).
   f. Student Indicates Lack of Understanding (ILU).
   g. Student Reads (Read).
   h. Student Refers (SRefer).
   i. Student Sets Goal (SSG).
   j. Student Types (Type).
2. The student asks for help from the tutor.
   a. Assist Plan Assertion (APA).

   b. Assist Plan Question (APQ).
   c. Assist Understanding (AU).
   d. Student Informational Request (SIR).
3. The student indicates that the tutor's utterances were understood.
   a. Student Confirmation (SC).
4. The student checks the current answer.
   a. Student Simulates Process (SSP).
5. Miscellaneous non-task-related utterances.
   a. Student Comment (Comm).

The categories of tutorial actions in the student–tutor discourse are:

1. The tutor performs a portion of the problem solving.
   a. Tutor Example (TE).
   b. Tutor Focuses Attention (TFA).
   c. Tutor Reads (Read).
   d. Tutor Refers (TRefer).
   e. Tutor Types (Type).
2. The tutor offers guidance for the student's ongoing problem solving.
   a. Tutor Confidence Builder (CB).
   b. Tutor Hint (Hint).
   c. Tutor Indicates Difficulty (TID).
   d. Tutor Sets Goal (TSG).
   e. Tutor Supportive Statement (SS).
3. The tutor confirms a student step (TCS).
   a. Tutor Confirmation (TC).
   b. Tutor Elaboration (TElab).
4. The tutor gives error feedback after an incorrect student step.
   a. Tutor Correction (TCr).
   b. Tutor Plan-Based Feedback (PBF).
   c. Tutor Surface-Feature Feedback (SFF).
5. The tutor attempts to assess the student's understanding of a topic.
   a. Tutor Probe (Probe).
   b. Tutor Prompt (Prompt).
6. The tutor helps the student check the current answer.
   a. Tutor Simulates Process (TSP).
7. Miscellaneous non-task-related utterances.
   a. Tutor Comment (Comm).

The coding scheme was designed to depend on the content of the utterance rather than on the form of the speech act used. Although the choice of speech act could affect the way a student responds to an utterance (Graesser et al., 1993; Lepper et al., 1990), we viewed the content of the utterance as most representative of the problem-solving state of the student and tutor. For our analyses of tutorial

responses to problem-solving situations, we wanted to focus on the information communicated rather than on the style in which it was expressed. This information defines the situations to which the tutors were responding. Thus, "No—put a quote there." and "Don't you need a quote there?" are both categorized as Tutor Corrections, because each conveys the same information, namely that the student needs a quotation mark in the program.

We categorized each segment in only one category, because more powerful analyses are possible for mutually exclusive categories (Bakeman & Gottman, 1986). Example 2, shown earlier, displayed an interaction after both segmentation and categorization. Initially in Example 2, the tutor pointed the student to an example in the text. This is a Tutor Focuses Attention event. The student responded affirmatively to this point, offering a Student Confirmation. Soon thereafter, the tutor worked through the solution as the computer would, an action that falls into the category Tutor Simulates Process. After this Tutor Simulates Process, the student offered a confirmation, and the tutor began a discussion of how LISP treats function parameters.

All protocols were independently categorized by two of us (D. Merrill and S. Merrill). The rules we followed are presented in Appendix A. After completing all categorization, we examined coding reliability using Cohen's kappa (Bakeman & Gottman, 1986; J. Cohen, 1960). Kappa captures the agreement among multiple coders but adjusts the resulting value for the amount of agreement between coders that would be expected due to chance. A value of kappa greater than .70 is considered indicative of a reliable coding scheme (Bakeman & Gottman, 1986). The categorization in this study was highly reliable, with kappa = .81.

Our categorization scheme based on segmentation by idea does have potential limitations. Recall, for example, that each utterance is categorized as one event. It seems possible, however, that an utterance could in fact serve several conversational goals. For example, an utterance classified as a Tutor Elaboration could also contain a Tutor Confirmation: "Yes, right, you remembered that *last* returns a list." Because there is no explicit evidence of a change of topic, these would be segmented together and coded as one utterance, even though the utterance appears to serve two pedagogical goals simultaneously: to offer a confirmation and to elaborate on information present in the solution. In the analysis presented shortly, we take the overlapping goals of different categories into account by merging certain categories that overlap significantly.

This limitation did not appear to interfere with categorization. Our results indicate that it was in fact possible to assign each utterance to a single category with high reliability. Thus, constraining each utterance to only one category appears to be a reasonable principle for categorizing these problem-solving events.

In addition to identifying each problem-solving event, we also identified those actions that contained an erroneous assertion or solution component. Errors play a critical role in learning. For example, explaining why an error occurred may help novices avoid the error in the future and highlight areas of confusion (Chi,

Bassok, Lewis, Reimann, & Glaser, 1989; Schank & Leake, 1989). Errors can also lead to serious floundering, however, potentially interfering with learning (Anderson, Boyle, & Reiser, 1985; Lewis & Anderson, 1985; Reiser, Beekelaar, Tyle, & Merrill, 1991). Tutorial assistance provided for locating and repairing errors has become a focus of research in both human and computer-based tutoring (e.g., Anderson et al., 1985; Littman et al., 1990; McArthur et al., 1990; Reiser, Kimberg, Lovett, & Ranney, 1992). Thus, errors represent particularly critical events around which to focus our analysis of tutorial strategies and the associated learning outcomes. To achieve this goal, we located and categorized every error and then identified the utterance that indicated an error had occurred, according to the following procedures.

Again, two of us independently looked through all the utterances and computer interactions to locate student errors. Errors included student goals that were not needed in the problem, required goals that were forgotten, incorrect assertions about functions or concepts, and syntactic mistakes, as well as slips such as typographical mistakes. In contrast to the categorization of conversational events, in this analysis we considered the speech act of the utterance. We did not categorize questions as errors, because questions are explicit requests for information rather than situations in which a student asserts something to be true that is false. Thus, Utterance 1 in Example 3 was marked as an error because the assertion about *append* is false, but Utterance 2 was not categorized as an error even though the fact proposed is incorrect.

Example 3:

1. OK, *append*, *append*, let's see, that's the one that takes an atom and a list . . .
2. Is *append* the one that takes two atoms?

Focusing on nonquestion mistakes enables us to see precisely what tutors do when students make a mistake rather than what happens following an explicit request for information. We were able to identify the errors reliably, kappa = .75. There were 1,242 errors in the problem-solving sessions, or approximately 25 per hour. More errors occurred during the third chapter, as the material became more difficult, but substantial numbers of errors occurred during the first and second chapters as well.

After identifying the erroneous actions, we categorized each one. This categorization of errors was designed to determine if tutors responded differently to errors of varying types. Accordingly, our error categories captured mistakes that have been discussed as central for learning, such as errors in the syntax of solutions (Anderson et al., 1985; Reiser et al., 1991), confusions about the semantics of basic operators (Anderson, 1989; Reiser et al., 1992), problems with goal structures (McArthur et al., 1990; Singley, 1990; Soloway, 1986; VanLehn, 1990), and other errors that one would expect to occur in such a task, such as typographical errors. We originally designed nine error categories, including four

TABLE 1
A Listing of Each Error Category Used,
Its Frequency of Occurrence, and Its Definition

Error Type	*Frequency*	*Category Definition*
Typographical	360	A typing error, involving a misspelling or an illegal keystroke.
Syntactic	435	The addition of an unneeded parenthesis or quotation mark or the deletion of one that is needed.
Semantic		
Operator	123	Asserting that a function does something it does not do or attempting to apply a function when it cannot be applied.
Concept	59	An error relating to the concepts: atom, list, *nil*, variable, or elements (of a list).
Goal		
Incorrect goal	178	Stating a goal to achieve that is not needed in the problem or will not help the student solve the problem.
Skipped goal	69	Skipping a goal that is needed in the problem. This could occur when setting up an initial goal structure or when solving the problem, and requires explicit evidence that the student has failed to achieve some subgoal.

slightly differing variants of one error group called *semantic errors*. In fact, we found occurrences of only two of these four variants among the student errors, so we discarded the two empty categories. We also initially considered "dead code," a situation in which extra functions are left in a solution but do not affect it in any way, as a potential error type. The tutors never responded to these situations, however, and the students' solutions actually produced the correct result, so we did not consider these 18 cases in our analyses of student errors. The remaining six error categories shown in Table 1 included 1,224 errors. Once again, two of us independently categorized the errors. The categorization was reliable, kappa = .70.

Finally, after locating the errors, we looked to see which participant indicated that an error had occurred and how the indication was performed. For example, the student might make an error and then notice it and begin a repair. In Example 4, the student made a syntactic error in the Else clause of a conditional by putting two parentheses instead of just one before the *t* and then flagged the error herself.

Example 4:

Student: [typing]
(defun classify (arg)
(cond ((numberp arg) 'number)
((null arg) 'nil)
((t
Student: Whoops, I don't need that many.
Tutor: Right, exactly. You caught yourself.

Alternatively, the tutor might comment on the error, as in Example 5, which occurred earlier in the same problem.

Example 5:

Student: [typing]
(defun classify (arg)
(cond ((numberp arg) number
Tutor: Now actually, um, for number, you want the actual word.
Student: So I have to put this [a quotation mark].
Tutor: Yes, you have to put a quote.
Student: [typing]
<DEL> . . . <DEL>*'number)*

The student's self-initiated correction in Example 4 and the tutor's comment in Example 5 indicated that an error had occurred. We call such utterances *error flags*. We classified an utterance as an error flag if it indicated that a problem-solving event was incorrect. Error flags ranged from very specific, as in Example 5, to very general, such as the tutor saying "Look back up there—there might be a problem." General utterances alerted the student that one of the recent steps contained an error but did not tell which step was wrong.

We did not restrict the categories of utterances that could serve as error flags. Of course, given the nature of some of the category definitions, certain categories, such as Tutor Confirm Step, were not used to indicate that an error had occurred. Thus, although all categories could serve as possible flags, only a subset were actually used as error flags. Assist Plan Assertion was the most common error flag from students. Tutor Error Feedback was the most common tutor error flag, followed by Tutor Prompt. Two of the authors reliably marked the flag for each of the 1,224 errors, kappa = .75.

Errors did not always result in an incorrect solution attempt. In some cases, students made an error in a step but immediately located it and began the repair within the same event, as in Example 6.

Example 6:

Student: [typing] *(my-or a*<DEL><DEL>*'a*

This student did not put a required quotation mark before the constant *a*, but she immediately repaired the error without assistance. We marked this as an error with an immediate flag by the student. Even though the student needed no help in this case, she experienced an impasse that had to be overcome. To account for all impasses and their repairs, we included these cases in the analyses as well. Furthermore, in some of these cases in which the student repaired his or her own error, there was no explicit utterance that marked the error. Instead, the self-correction behavior was considered as both flagging and repairing the error.

In many cases, the error flag only initiated the error repair process, which could require several events to complete. To determine how many events were required to repair errors, two of us independently located the event that achieved the repair for each error. The repair sometimes occurred within the same event, as in Example 6, or otherwise occurred a few events later. Finding the repair location, given the location of the error itself, was done reliably, kappa = .90.

Having described the categorization of each utterance, the finding of all errors, the identification of utterances beginning an error recovery process, and the location of the end of each error episode, we next turn to analyzing these data.

RESULTS AND DISCUSSION

Before describing the tutors' methods of assisting the students during the learning sessions, we must demonstrate that the tutors were in fact effective. To do so, we analyzed the students' posttests and the duration of the learning sessions. Recall that all students, regardless of condition, worked on all the problems until they were all correctly solved. The tutored students completed the material in just over half the time that the nontutored students required (300 min vs. 550 min), $F(1, 13) = 24.5$, $p < .01$. There were no differences between the groups' performance on the posttest. Students in the one-on-one tutoring condition received 97% of the points possible, and the independent problem-solving students scored 95% on average. This lack of posttest differences does not indicate a lack of pedagogical effectiveness for tutors, because one would expect that solving all the problems correctly should lead to significant domain mastery (Anderson & Corbett, 1993; Newell, 1990). Thus, it is not surprising that both groups were able to achieve the same degree of understanding of LISP. The important difference is the time it took to do so: The tutored students achieved equivalent domain mastery, in spite of spending substantially less time on task than the independent problem-solving students. Thus, the tutors did provide clear cognitive benefits, so we next examine the protocol data to specify the actions performed by the tutors and the situations in which they occurred.

In this section, we present our analyses of the approximately 15,000 student–tutor interactions over the 50 hr of sessions to describe the ways tutors assist students in developing domain mastery. We present a model to show why the assistance should be helpful. Examples 1 and 2 demonstrated the complexity of the student–tutor interactions, including confirmations, corrections, and goals-setting. Our fine-grained identification of all student actions and tutorial responses in extended problem-solving sessions allows us to investigate how tutors support and guide students' problem solving.

Domain mastery typically begins by studying expository text and annotated examples (Chi et al., 1989; Faries, 1991; Gentner, 1983; Gick & Holyoak, 1980; Pirolli, 1991; VanLehn, Jones, & Chi, 1992). Elaboration of declarative material

via questioning, predicting, and explaining can facilitate solving problems later (Chi et al., 1989; Graesser, 1992). Learning also takes place at the critical point when students attempt to apply declarative knowledge gained from the text and examples to solve new problems (Anderson, 1983, 1987; Trafton & Reiser, 1993a). We are concerned with the overall development of domain expertise in these learning sessions rather than the differential roles of solving problems and understanding expository materials. Thus, we consider student events occurring either while reading or while working on assigned exercises as problem-solving actions and focus on the role of these events in learning.

Protocol analyses typically make use of new categories created by combining the originally coded events (Bakeman & Gottman, 1986). Often the original categories are coded at an extremely fine grain, and then clusters of events taken together represent functional groups. For example, several of the original student events taken in combination describe the problem-solving process. Individually, these student events capture different actions that could take place during problem solving, such as assertions and elaborations made while reading the text, generating or studying a concrete example, setting goals, or creating new LISP expressions. All of these are categories in our coding scheme and were observed in our study. However, the fine distinctions between them are not relevant for understanding how the tutors assisted overall problem solving, including encoding the text and examples. Thus, we combined these categories into a new category called *Student Problem-Solving Action*, which is used throughout the succeeding analyses.

In addition, we created another higher level category, *Tutor Confirm Step*. Recall that earlier, we pointed out that segmentation by idea often led to Tutor Elaborations, including explicit confirmations. Because all segmentation was performed before categorizing any utterance, we could not go back and break out the confirmatory portion from the elaborative portion of a Tutor Elaboration that contained a confirmation. Thus, we decided to include Tutor Elaborations containing explicit confirmations in Tutor Confirm Step. Furthermore, elaborating on a student assertion usually implies an implicit confirmation, because the new information is added to the tacitly agreed-to student action. Therefore, we defined the category Tutor Confirm Step to include utterances originally categorized as Tutor Confirmations as well as utterances originally coded as Tutor Elaborations, thereby focusing on the role of confirmatory feedback in problem solving.

Figure 1 shows the events in the tutoring sessions, displaying the results of our categorization of the data. Each object in Figure 1 is a type of event. The half circles are tutor events such as Tutor Confirm Step, in which the tutor offered confirmatory feedback to the student. The squares within circles are student events, such as Student Problem-Solving Action, the category that captured the actions while students solved problems and tried to interpret the text.

In Figure 1, the arrows represent the most important source of information. They show that some event type followed some other event type. To construct the figure, we examined the frequency with which each event type followed all

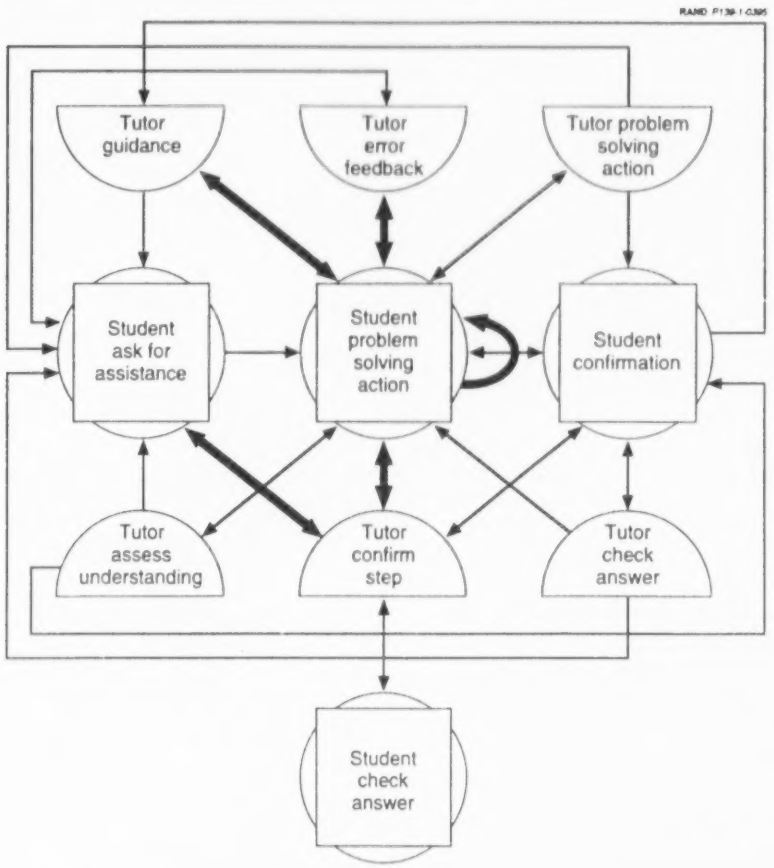

FIGURE 1 A presentation of student and tutor actions and their chronological relations. The half circles represent tutor events, and the squares within circles represent student events. The arrows connecting objects describe which events followed other events in the data. Heavier links indicate high-frequency transitions that occurred more than 100 times.

other types of events. We then examined this transition analysis (Bakeman & Gottman, 1986; Fisher, 1991) for the most common chronological sequences, enabling us to specify precisely what sorts of events occurred and how they were related during the sessions. The arrows represent all the transitions between one event and another that occurred more often than 10 times in the protocols. The wide arrows are the most frequent transitions in the data, those that occurred more than 100 times. For example, Tutor Confirm Step often followed Student Problem-Solving Actions. For completeness, the entire transition matrix is provided in Appendix C.

In addition to the combined categories Student Problem-Solving Actions and Tutor Confirm Step, Figure 1 contains a category called Tutor Error Feedback. As described earlier, the manner in which tutors respond to errors can be crucial for learning. For the initial picture of the tutoring sessions shown in Figure 1, we merged three categories of explicit error feedback (Tutor Correction, Tutor Surface-Feature Feedback, and Tutor Plan-Based Feedback) into Tutor Error Feedback. The categories included in Tutor Error Feedback contain only tutorial utterances that provide direct and explicit guidance after errors, but these events do not exhaust all possible tutorial responses to student errors. Any utterance could, in principle, be used in response to an error.

Because we believe that most learning occurs during the sort of events captured in Student Problem-Solving Action (Anderson, 1983; Laird et al., 1986; VanLehn, 1988), we focus on this category in the remainder of this article. Specifically, we present two sorts of analyses of tutorial assistance. First, we discuss the means by which tutors help keep the students' problem solving on productive paths via confirmatory feedback, error feedback, and other guidance. We then turn to a finer examination of errors to uncover the ways tutors offer feedback, to see if tutors in fact respond differently, depending on the nature of the student's error, and to examine how the student and tutor work together to repair errors.

How Tutors Keep Problem Solving Productive

In this section, we examine the strategies tutors use to provide guidance to students while they are in the process of understanding and solving problems—that is, engaged in Student Problem-Solving Actions. This section considers the ways tutors help keep student problem solving productive and continuing. We look initially at correct problem-solving actions to see how tutors respond and then turn to responses following impasses and errors.

Confirmatory feedback. How do tutors respond when students make correct problem-solving actions? Fox (1991) argued that tutors offer confirmatory feedback after correct steps. Our category Tutor Confirm Step captured this sort of tutorial feedback. Of the 3,506 Student Problem-Solving Actions in our data, 1,495 (44%) were followed immediately by a Tutor Confirm Step. This information is represented by the wide arrow from Student Problem-Solving Action to Tutor Confirm Step in Figure 1. The remaining 56% of transitions from Student Problem-Solving Action consisted primarily of occurrences of one Student Problem-Solving Action following another and Tutor Guidance utterances following a Student Problem-Solving Action.

Notice that the 44% was calculated using all Student Problem-Solving Actions, including erroneous steps. When considering only the 2,261 correct problem-solving actions, the picture becomes even more striking, with 66% of correct Student Problem-Solving Actions receiving confirmatory feedback. Almost no

incorrect steps received confirmatory feedback. Thus, although tutors commonly confirm correct actions, they appeared to be careful not to offer confirmatory feedback after erroneous actions.

These problem-solving steps were not, by and large, complete solutions. This high proportion of confirmations did not reflect situations in which the student had created an entire solution to which the tutor responded, "Yes." Most solutions required multiple problem-solving actions, even when no errors occurred. In fact, problems that called for definitions of new LISP functions, as in the second and third chapters, required an average of 10 correct events to complete, even with no erroneous events. These problems received an average of 6 Tutor Confirm Steps per problem as well. Thus, tutors offered confirmations very often—for 66% of correct events—and these confirmations occurred during ongoing problem solving, rather than at its successful completion.

To emphasize further that students received these confirmations during problem solving, note that 43% of Tutor Confirm Steps were followed immediately by another Student Problem-Solving Action. Example 7 gives an example of the use of Tutor Confirm Steps during attempts to solve a problem called *first-elem*. The student initially set up the first part of the expression to be typed, and the tutor responded with a confirmation (Tutor Confirm Step). The tutor offers further confirmations as problem solving continues through the problem.

Example 7 (examples of Tutor Confirmations in ongoing problem solving):

1.	SPSA	*Student:*	So, you do defun, um, first-elem.
2.	SPSA	*Student:*	[typing] *(defun first-elem*
3.	TCS	*Tutor:*	Right, that's the function name.
4.	TID	*Tutor:*	Now comes the tough part.
5.	SPSA	*Student:*	Now comes the parameters.
6.	TCS	*Tutor:*	The parameter list, right.
7.	SPSA	*Student:*	So, it just needs to have a list.
8.	TCS	*Tutor:*	Um hum.
9.	SPSA	*Student:*	It would just be *list.*
10.	SPSA	*Student:*	[types] *(list)*
11.	TCS	*Tutor:*	Sure.

It might be suggested that these confirmations are in fact simply conversational requirements, because people are expected to respond to others' statements (Grice, 1975). If this were the case, tutors would have confirmed all steps. Tutors did not confirm all steps, however, but offered confirmations only in response to correct steps and responded to errors with other types of tutorial actions. Thus, tutor confirmatory feedback was informative and told the student that the previous action fell onto a profitable solution path.

Tutors could also encourage students to continue on the current, productive solution path via Tutor Guidance. A tutor response such as "So next we need

the Else case" includes an implicit confirmation of the previous step. In other words, the tutor is basically saying, "OK so far, now the Else case is next." Tutor Guidance could include utterances that were motivational in nature, such as "Yeah, you're doing just fine," which also encouraged the student to continue along the correct path. Tutor Guidance events made up another 16% of events subsequent to the 3,506 Student Problem-Solving Actions and an additional 70% of events following correct Student Problem-Solving Actions. Thus, Tutor Guidance utterances that contained an implicit confirmation represent another important means by which tutors kept problem solving productive by encouraging students to continue on a productive solution path.

These results support the claims of Fox (1991). They show that tutors do offer confirmatory feedback throughout the problem-solving process. Correct events usually receive confirmations immediately, and incorrect events do not receive confirmatory feedback. When students are on a promising solution path, this type of encouragement appears to be one method by which tutors help guide students' problem solving. We next turn to the tutorial responses that followed errors.

Responses to incorrect problem-solving steps. Not all solution steps are correct. Errors offer a particularly crucial opportunity for learning. Some researchers have argued that recovering from errors carries great potential dangers (Anderson, 1983; Lewis & Anderson, 1985; Sweller, 1988), whereas others have argued that recovering from and explaining impasses is the key to effective learning (Chi et al., 1989; Laird et al., 1986; Schank & Leake, 1989; VanLehn, 1990). First, we consider how errors are uncovered in the problem-solving sessions. How do tutors help students recover from errors? Do tutors allow students to locate and repair their own errors, a strategy emphasized by some learning researchers (Papert, 1980; Schank & Leake, 1989), or do tutors find the errors for the students?

Errors were not left unnoticed for very long. In fact, 75% of all errors in the sessions were indicated within 2 events. Typically, an error occurred, the student or tutor made one other utterance, and then the error was flagged. Recall that an error flag is the initial indication in the discourse that an error has occurred, and the flag could be any of the different types of events we coded, such as a Tutor Focuses Attention: "Umm, look back up there—there might be a problem." Thus, these problem-solving sessions were not typified by long exploratory searches during which errors might occur and not be noticed immediately, a pedagogical approach sometimes advocated (e.g., Papert, 1980). Example 8 shows a typical instance of a student flagging her own error, and Example 9 shows a tutor flagging a student error.

Example 8 (a typical example of an error flagged by the student):

1. SPSA *Student:* [typing] *palp (a b c c b a))*
[error: there needs to be a quotation mark before the list]

BEST COPY AVAILABLE

2. SCr	*Student:*	Oh, I didn't put the, uh //
		[flag for previous step]
3. TElab	*Tutor:*	Quote.
4. SC	*Student:*	Quotes.
5. TCS	*Tutor:*	Right.
6. SPSA	*Student:*	[types repair]

Example 9 (an example of the tutor flagging a student error):

1. SPSA	*Student:*	[Pause.] So you could use *or*. [unintel.] *Cond*. [Pause.] Um, *equal* [pause] *arg1*, true [pause]
		[error: the argument may not be equal to true]
2. SFF	*Tutor:*	Right. But what if it was, like, the atom *dog*. That counts as true=
		[flag for error in Step 1]
3. SC	*Student:*	=Uh-huh.

Because error repair was begun very rapidly, we next turn to the question of who noticed the errors. Fox (1991) and Lepper et al. (1990) argued that the student plays a major role in locating and repairing errors. In fact, of the 1,224 errors identified for analysis, almost half (47%) were noticed by the student. What sorts of errors did the students catch? Interestingly, most of the errors caught by the students (91%) were typographical errors or syntactically incorrect LISP expressions. Although there may be pedagogical advantages for students to find and repair many different sorts of errors, including errors involving problem goals, for example, students generally did not do so. Either they simply could not find these errors or our tutors did not allow them the leeway to do so. Next we consider how tutors responded to student errors.

Tutors flagged approximately 53% of all errors. Of the errors flagged by the tutor, only 41% were typographical or syntactic errors. The remaining 59% were errors relating to goals and to the meanings of LISP operators. Thus, tutors mainly caught the more difficult errors, whereas students mainly caught the low-level errors.

Figure 2 shows the number of errors flagged by the student and the tutor as a function of the number of events intervening between the error and the repair. Most student-flagged errors were caught very quickly—often on the same event, with a lag of zero—whereas many of the tutor-flagged events were more removed from the impasse. Most, but not all, of the tutor flags occurred very quickly after the error, usually on the next event. Thus, tutors flagged most of the more serious errors related to goals and to the meanings of LISP operators and generally did not allow students to find and repair their own errors, because the tutors commented on most errors immediately after the error occurred if the student did not notice it.

Analysis of the events following the error flags suggests that the tutors' error flags were used during the students' ongoing problem solving. Sixty-three percent

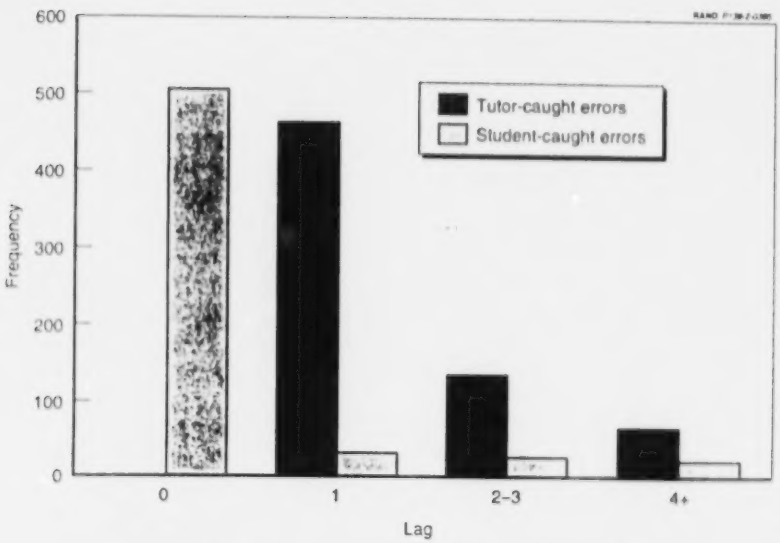

FIGURE 2 The number of events that intervened between a student error and the flagging of that error by the student or the tutor. A lag of zero indicates that the error was flagged during the same event within which it occurred.

of tutor error flags were followed immediately by another Student Problem-Solving Action, thereby indicating that students received the assistance and began implementing a repair. Almost all the remaining tutor error flags were followed by Student Confirmations. Although some have suggested that students have great difficulty understanding error feedback given by instructors (Graesser et al., 1993; Moore & Swartout, 1989), the students in this study seemed to understand the error flags very well, because most tutor error flags were not followed by requests for clarification and elaboration. Thus, error flags, like confirmations, are important to the ongoing problem-solving process.

These analyses have suggested that tutors did not allow students a great deal of time to discover their errors; if the student did not comment on an error almost immediately after it happened, the tutor did. Despite the potential benefits of students finding and repairing their own errors, students found mostly syntactic low-level errors. Repair of the more complex errors was at least initiated by the tutors. In subsequent analyses to be presented shortly, we consider the type of guidance tutors provided on errors and how the error recovery episode progressed.

Summary of tutorial guidance. In this section, we considered how tutors offered interactive support for student problem solving. We showed that tutors offered rapid confirmations and supportive guidance to correct student actions, even small actions such as interpreting a short text section or creating individual

components of a solution. These confirmations were more than the conversational politeness of acknowledging the other speaker, because the tutors offered confirmations only to correct steps. In contrast, incorrect steps were flagged very rapidly. Students caught almost half the errors, but the errors they noticed were primarily low-level errors such as typing mistakes or syntactic problems. The remainder of the errors were caught by the tutor, usually within 1 or 2 events. Thus, tutors were very much involved in the ongoing problem solving, because most student steps received some sort of tutorial response that helped guide problem solving, either a confirmation or notification of an error.

These analyses support a view of tutoring as guided learning by doing. When a student solved problems alone, he or she may have often had difficulty determining whether a step was correct or not. When a tutored student made a correct step, however, the tutor intervened to say it was correct, thus helping problem solving to continue. Also, when a student working alone made an error, he or she may not have noticed it for some time, making repair difficult and floundering likely. Tutors ensured that any errors were noticed very quickly, jumping in to tell the student that the step was incorrect, if needed. These confirmations and error flags served to guide the ongoing problem solving and keep it productive.

Having focused on the ways tutors help keep ongoing problem solving productive, we next turn to a more precise examination of the errors made during the learning sessions, focusing on the type of assistance tutors provided when they offered feedback. We describe the ways tutors and students worked together to repair errors, paying particular attention to the ways the tutors and students dealt with different types of errors.

Errors and the Content of Feedback

Considerable research has focused on the manner in which tutors scaffold students' recovery from errors and impasses. For example, Fox (1991) and Lepper et al. (1990) argued that tutors attempt to indicate errors to the student subtly, so that the student can perform the repair, whereas Littman et al. (1990) argued that tutors offer explicit error feedback. The content and style of error feedback can have significant effects on learning and motivation (Lepper et al., 1990; McKendree, 1990; Reiser, Copen, Ranney, Hamid, & Kimberg, in press), and, thus, understanding how tutors respond to students' errors can cast further light on how tutors guide their students.

Recovering from an impasse or error entails several components (Merrill et al., 1992). The first stage requires realizing that an error has occurred. Then, the erroneous features of the solution must be located, and the erroneous portion must be replaced with a successful fulfillment of the goal. Finally, it may be helpful, although not necessary, to understand why the error occurred. These components are fundamentally distinct, even though they usually occur as a group. A student might perform all the components, thereby completing all the

error recovery. At the other extreme, a tutor might tell a student exactly what went wrong and how to repair it. The error recovery process could also be a collaborative enterprise, with the tutor scaffolding the recovery as needed to get problem solving back on track but allowing the student to perform much of the work. If the tutor's main goal is to get the problem solving back on track, we would expect tutors to perform most of the error recovery components for the students through feedback. Alternatively, if errors are used as opportunities for learning, tutors might allow students to perform most, if not all, of this process.

In this section, we investigate whether there are any regularities in tutorial responses to errors. We first define the types of errors students committed and then present the different types of error feedback tutors used to initiate error recovery in our categorization scheme. Next, we examine the relation between tutor error initiations and error type. In the final part of this section, we discuss how students and tutors collaborated to repair errors.

Before describing tutorial responses to errors, we first describe the types of errors in detail. Recall that two independent coders identified all errors in the verbalizations and typing (see Table 1). Some of these errors represent difficulties in planning a solution, others involve incorrect assertions about functions and concepts in LISP, and still others reflect difficulties in implementing a correct solution. These three categories of errors capture a continuum of behavior ranging from planning difficulties to problems implementing a solution. Thus, we can determine whether tutors respond differently to errors where the repairs have differing severity. The present analysis excludes the 360 typographical errors that were typically slips and self-corrected by the student. We focus on the remaining categories of errors. These five categories form three groups that we use as the basis for this analysis: syntactic, semantic, and goal errors.

Syntactic errors are difficulties in communicating an expression to the LISP interpreter correctly so that the expression can be understood. This category covers cases where the programming constructs and algorithms used would achieve the stated goal, but the student implemented a construct incorrectly in the language. These errors consisted only of the addition of extra parentheses, the deletion of needed parentheses, or the addition or deletion of quotation marks.

The second class, *semantic errors*, consists of inappropriate uses of LISP constructs and includes the subtypes *operator* and *concept* errors, as shown in Table 1. These covered expressions that were syntactically correct but made use of inappropriate constructs. One way of misapplying a construct was to apply a function when the preconditions of the function were not met. An example of a precondition failure occurred when a student typed *(cons 'a 'b)* to construct a list; in this curriculum, *cons* requires a list as its second argument, but the student used an atom. To consider another example, the student might claim incorrect output of a function that could be applied to the data. For example, one student said "*(member 'b '(a b c d))* returns *(b)*," when in fact it would return the list *(b c d)*. Thus, although *member* can be applied in this situation, the student did

not apply it correctly. These semantic errors consisted of more than simply a failure in communicating a solution to the computer; they are erroneous choices or applications of that which must be communicated: operators in the domain.

The third category, *goal errors*, consists of errors concerning setting or fulfilling goals. These errors included the subtypes *incorrect goal* and *skipped goal* (see Table 1). In these cases, a solution was syntactically correct and used the correct function for a goal, but the goal under consideration was in some way incorrect. When reading the problem, the student must try to set up an initial goal structure to begin solving the problem. The student might have difficulty or make mistakes doing this, as reflected by the student who said, "so I just return the [first] variable," when the correct subgoal was to return a list of the first and second variables. In other cases, students were solving the problem, choosing and achieving subgoals, and failed to implement a subgoal that had been previously set. In these cases, students were able to set up a goal structure but then failed to achieve a goal that had been present in the structure at the start. These errors concern an even more critical component of problem solving than semantic errors. They involve planning a solution apart from its implementation.

Having presented the three classes of student errors, we can now consider how tutors responded to each type. First, we consider the events used to initiate error recovery. Following that analysis, we investigate how the type of feedback was related to the type of error. Recall that Figure 1 included a category called Tutor Error Feedback. This category is composed of three categories: Tutor Correction, Tutor Surface-Feature Feedback, and Tutor Plan-Based Feedback. These differ in the portions of the error recovery process initially performed by the tutor. When tutors intervene, they must determine how much of the error recovery process they will perform. Analyzing the occurrence of tutorial error feedback will cast light on how tutorial feedback is tailored to student errors. First, we review the three categories of tutorial feedback.

We defined *error feedback* as utterances that contained a reference to incorrect features of the student's solution. The feedback might also contain information about how to repair the error. Although there was usually only one error feedback utterance per error, our coding scheme allowed for multiple feedback utterances per error. Thus, our definition of error feedback is very similar to our definition of *error flag*, except that a flag might not point to any particular feature of the solution, whereas an instance of error feedback must point to a feature. Tutors used one of the error feedback categories to flag errors in 95% of cases. However, because we are concerned here with the information conveyed by the tutor in response to a student error, in this analysis we examine the tutorial error feedback instead of error flags. As shown later, tutors could indicate features at differing levels of abstraction and with differing amounts of information about the repair.

We categorized feedback as Tutor Correction if the tutor responded to a student error by telling the student what the correct action should have been and how to repair the error. Thus, this category captures tutorial utterances that perform

all the components of the error recovery. A Tutor Correction might contain an explanation of why the step was incorrect, or it might simply be a directive. For example, one tutor said, "You want to quote that, since it'll be a function call otherwise," after the student typed *(listp (a b c d))*. This told the student that a quotation mark before the *(a b c d)* had been forgotten and why that mattered. In a different situation, a tutor said, "You'll need to use *and* there, instead of *or*." These utterances differed with respect to how much explanation was given along with the correction, but both were Tutor Corrections, because each told the student exactly how to repair the impasse.

We also had two categories for error feedback that initially provided fewer of the error recovery components. An utterance was considered Tutor Surface-Feature Feedback if it only pointed out an erroneous feature explicitly present in the student's solution. For example, instead of offering a Tutor Correction to the *listp* example just quoted, the tutor could have offered a Tutor Surface-Feature Feedback by saying, "An unquoted list is a function call." This type of feedback points out to the student where the problem is and often includes information about which component is incorrect but does not directly suggest a repair. The repair may be easily inferable from the feedback, as in the last example in which the repair is to add a quotation mark, but an inference is required, nonetheless. For example, the students read in the textbook that *cons* took two arguments, an atom as the first argument and a list as the second. When a student began typing an expression designed to rotate the last element of a list to the front using the function *cons* and typed *(cons (last lis)*, the tutor intervened to say, "*Last* returns a list, not an atom." The student had to infer what the tutor meant by the feedback. This feedback could indicate that *last* or *cons* was the wrong function to use, that the student had forgotten some additional function that needed to be used, or even that the *last* should have been the second argument to *cons* instead of the first. All these inferences were potentially intended in the tutorial feedback. The feedback itself served to make the student aware of the general location of an error. The student had to infer the nature of the error from the feedback, set a goal for the repair, and begin replacing the errant portion of the solution. Thus, students had more opportunity to participate in the error repair after a Tutor Surface-Feature Feedback than after a Tutor Correction.

Finally, the tutor could leave even more of the error repair to the student. The category Tutor Plan-Based Feedback simply restated the goal the student should have been pursuing. For example, a tutor said, "The function should return *found* if the item is in the list," after a student coded a function that returned *t* in that case. This told the student that, although the programming constructs employed may have been used appropriately, a goal embodied by that portion of the program was incorrect. Once again, even though the inference required to determine which goal was incorrect may have been fairly straightforward, the student still had to make an inference. Furthermore, an utterance such as "You want to see if both arguments are lists, not if either one is a list" in response to the student code

(or (or (listp arg1) (listp arg2))) was not as clear. In fact, this student needed to replace the second *or* with an *and*. However, the student might sensibly infer that the correct action was to have only one *or*, for example, and therefore erroneously delete the first *or*. This type of feedback requires the student first to localize the difference between the goal the tutor states and the goal the student was pursuing and then replace the erroneous portion of the solution. Thus, the student has even more opportunity in this situation to participate in the error recovery process than in the two previous feedback types.

For this analysis, we considered only those 575 errors (95% of the tutor-caught errors) that received one of these types of explicit tutorial error feedback, excluding the 42 typographical errors caught by the tutor and the 33 errors that did not receive one of the three forms of explicit error feedback. We then examined the frequency with which each type of student error elicited each of the three types of tutorial response. This analysis, shown in Figure 3, reveals a strong relation between the nature of the student's error and the type of feedback provided by the tutor, $\chi^2(8, N = 500) > 150$, $p < .001$. The tutors exhibited a strong tendency to intervene with a different guidance strategy, depending on the nature of the error.

When the error consisted only of a low-level syntactic detail, the tutors prevented the students from floundering by suggesting exactly how to repair the error. Example 10 presents a typical example of this type of student error and

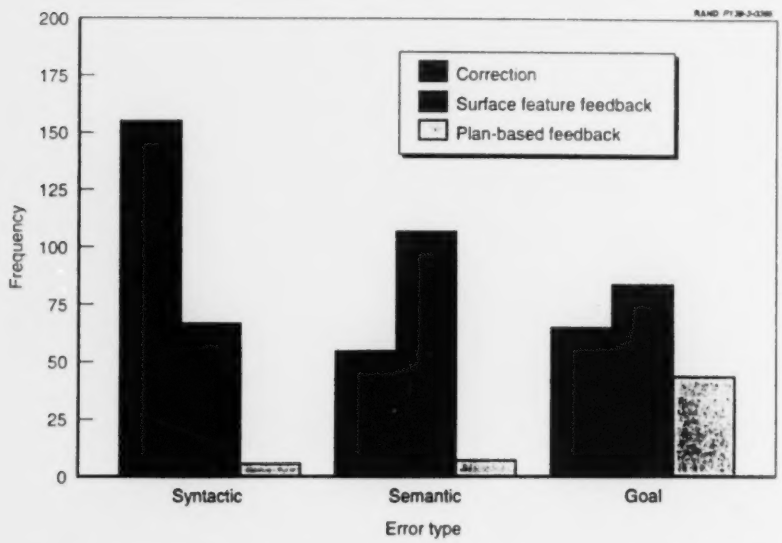

FIGURE 3 The frequency of tutorial feedback that occurred in response to each type of student error.

tutorial response episode. In Event 2 of this example, the tutor told the student to insert a quotation mark that the student had omitted.

Example 10 (an example of a Tutor Correction following a Syntactic Error):

1. SPSA	*Student:*	[typing] *(back (a b c)*
		[error: should have been *'(a b c)*]
2. TCr	*Tutor:*	Now remember to quote that=
	Student:	=Umm.=
		=argument. *(a b c).*
3. SPSA	*Student:*	[Begins typing repair]
		<DEL><DEL> . . . <DEL> *'(a b c))*

In contrast to syntactic errors, most semantic errors received Surface-Feature Feedback. In these cases, the tutor pointed out the erroneous feature of the solution to the student rather than suggesting an explicit repair. Example 11 presents a typical example of a tutor focusing on a feature of the student's solution in this manner. Here, in Event 6, the tutor described the correct behavior of *append*, indicating it should not be applied in this situation.

Example 11 (an example of a Tutor Surface-Feature Feedback following a semantic error):

1. SPSA	*Student:*	[types] *(defun numline (num)*
2. SPSA	*Student:*	So you're putting together two lists if [laugh]
3. TSG	*Tutor:*	If the first one is zero.
4. SPSA	*Student:*	If *number* is zero. And *nil* otherwise . . .
5. SPSA	*Student:*	[types] *(append*
		[error: *append* cannot be applied to nonlist arguments]
6. SFF	*Tutor:*	Remember *append*, what ap . . .
		What *append*'s arguments must be, though.
7. SPSA	*Student:*	Oh, two lists.
8. TCS	*Tutor:*	Right
9. Comm	*Student:*	So [laughs]
10. Prompt	*Tutor:*	'Cause if, if it comes out with *t*=
11. SPSA	*Student:*	=*t*, it's not going to be a list.
12. TCS	*Tutor:*	It's not a list, right.
13. SPSA	*Student:*	OK, then I just have to make, have to create a list.
14. SPSA	*Student:*	[types] <DEL><DEL> . . . <DEL>*list*
		[Error in Step 5 is repaired here.]

The third type of student error, goal errors, also received Tutor Corrections and Tutor Surface-Feature Feedback. In addition, however, this type of error elicited a substantial amount of Plan-Based Feedback, not often given in response to the other error types. Example 12 presents a typical example in which the tutor commented on the student's goals. Here, the student used an incorrect order

BEST COPY AVAILABLE

of conditional cases in the solution, and the tutor reminded the student of the importance of case order (in Event 2).

Example 12 (an example of a Tutor Plan-Based Feedback following a goal error):

1. SPSA	*Student:*	[typing] *(defun carlis (oldlist)* *(cond ((listp oldlist) (car oldlist* [error: this is not the first case]
2. PBF	*Tutor:*	eh um, a good, uh, kinda like a good thing to keep in mind then in ordering is to put the, most specific thing first . . . most specific test first.
3. SPSA	*Student:*	[Begins typing repair, deletes first case] *((null oldlist) 'nil)*
4. TCS	*Tutor:*	Right, this is a case where you do need two parens.
5. TCS		Um, it's kinda unique, because the first paren is the starting of the case, and then the second one is for the function that you are about to call.
6. SC	*Student:*	OK OK
7. TElab	*Tutor:*	If you call a function.

These results suggest an interesting relation between the type of error and the tutor's initial response to the error. In those cases when the error occurred while the student was trying to communicate a solution to the computer, the tutors did virtually all the error recovery process. If the error concerned the selection of an operator in the domain, however, the tutors pointed out the erroneous feature to the student and allowed the student to participate in the remainder of the error recovery. If the error was at an even higher level, constructing and maintaining the problem's goal structure, the tutor assisted with the subgoal selection by offering a Tutor Plan-Based Feedback, thus providing the student with the opportunity to identify the errors, plan the recovery, and execute it. In these cases, the tutor performed little of the recovery process, allowing the student to do most of the recovery.

These results suggest very clearly that the effectiveness of tutorial feedback may arise because of the contingency of feedback style and content on the nature of the student's error. One alternative possibility to be considered is whether the categories of tutorial response were defined so that each was logically possible only on a subset of error types. For example, perhaps tutors could offer Tutor Corrections only in response to syntactic errors.

In fact, however, tutors did offer all types of feedback to all types of errors, albeit with differing frequencies. To illustrate this point further, the following outline contains examples of tutorial feedback demonstrating a plausible tutorial response of each type to each type of student error. These examples are taken directly from our protocols, slightly modified so that each example refers to the same error, making it easier to compare the different feedback examples. The

original feedback was of the type presented in the outline (e.g., Plan-Based Feedback) and did refer to the same type of error (e.g., syntactic error), however. Notice that tutors were able to offer Plan-Based Feedback, even to a syntactic error (1c). This feedback refers to the goal the student should have been working on, thus allowing the student to identify and correct the error, but refers to the syntactic error of adding an undesired quotation mark before the variable *lis*. Thus, our three tutorial feedback categories are not biased in their definitions to restrict the feedback to particular error types. Instead, the pattern in the data appears to represent a real aspect of individualized instruction.

Examples of all three types of explicit error feedback for different error types are:

1. *Syntactic error:* Student typed *(member item 'lis)*, but there should not have been a quotation mark before the *lis*.
   a. *Tutor Correction:* "You should remove the quote before *lis*."
   b. *Tutor Surface-Feature Feedback:* "Remember, a quoted atom is treated literally, it's not evaluated."
   c. *Tutor Plan-Based Feedback:* "I think you wanted to look for *item* in the value of *lis*, not in the atom *lis*."
2. *Semantic error:* When attempting to get the element following some target item in a list, the student typed *(car (member item lis))*, apparently forgetting that *member* returns a list of *item* and all items following it in *lis*.
   a. *Tutor Correction:* "You want the *car* of the *cdr* of the return."
   b. *Tutor Surface-Feature Feedback:* "Remember *member* returns the tail of the list starting with the item."
   c. *Tutor Plan-Based Feedback:* "Didn't you want to get the element after *item*?"
3. *Goal error:* When trying to see if two items were both lists in the context of a larger problem, the student typed *(or (or (listp arg1) (listp arg2)))*, but the inside *or* should have been an *and*.
   a. *Tutor Correction:* "The second *or* should be an *and*."
   b. *Tutor Surface-Feature Feedback:* "You don't want two *or*'s, do you?"
   c. *Tutor Plan-Based Feedback:* "You meant to see if both things were lists, not if either one was a list."

In the next section, we detail the students' involvement in the error recovery process and highlight the differing outcomes of each type of tutorial feedback.

What Did Students Do in the Error Recovery?

We have focused so far on the role of tutorial guidance, but we have not yet considered the role that students play in the error repair. One possible scenario, given the active nature of the tutor's guidance, is that students play an active

role when solving problems but switch into a subsidiary role when errors occur, following the tutor's directions, perhaps asking questions to clarify advice (cf. Moore & Swartout, 1989), but not playing an active role in the recovery. In this section, we examine the role of students in the error recovery process.

In our analysis of tutorial feedback, we suggested that feedback varied in the portion of the error recovery process left to perform after the feedback. To confirm this, we analyzed the median number of events required to repair an error after it occurred. If more of the error recovery process remained after a Tutor Plan-Based Feedback than after a Tutor Correction, more events should be required to achieve the subgoal correctly after a Tutor Plan-Based Feedback. As shown in Figure 4, it took more events to repair a goal error (a median of 4 additional events) than a semantic error (3 events). Syntactic errors exhibited the shortest repairs, requiring a median of 2 events after the error occurred. These longer durations of repair episodes suggest increased difficulty forming a repair, consistent with our finding that the tutors left more of the error recovery process to be performed by the student.

Analysis of the events occurring during these error episodes reveals a collaborative repair process. Although errors were typically repaired very quickly, sometimes after as little as one event, the repair process did not typically consist of the tutor leading a passive student back on to a productive solution path. Instead, students attempting to repair an error proposed actions to take, and then the tutor and student worked together to get the solution back on track.

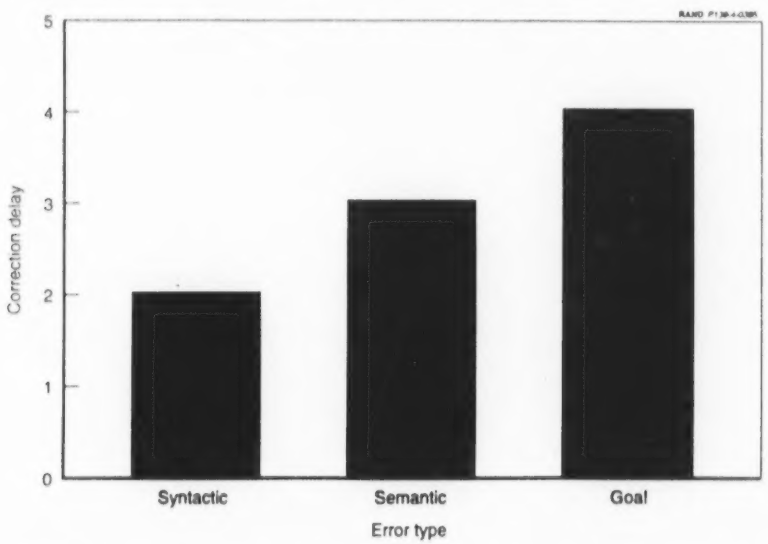

FIGURE 4 The median number of events required to repair each of the different types of errors. Longer repairs indicate more difficult or more collaborative repairs.

Figure 5 displays the common sequences of events following an error, those occurring more than 50 times in all protocols. The student error is in the upper left corner of the figure, and the next event was usually one of the types of tutor error feedback. Infrequently, there were a few student actions intervening between the error and the feedback. These are represented by the Other Student-Problem-Solving Actions box in the upper right-hand corner, with the dotted links representing infrequent events. The collaborative repair typically began with the Tutor Error Feedback. A Student Problem-Solving Action usually followed the feedback, presumably implementing a partial repair. The tutor often gave confirmatory feedback to the Student Problem-Solving Action, to which the student also offered

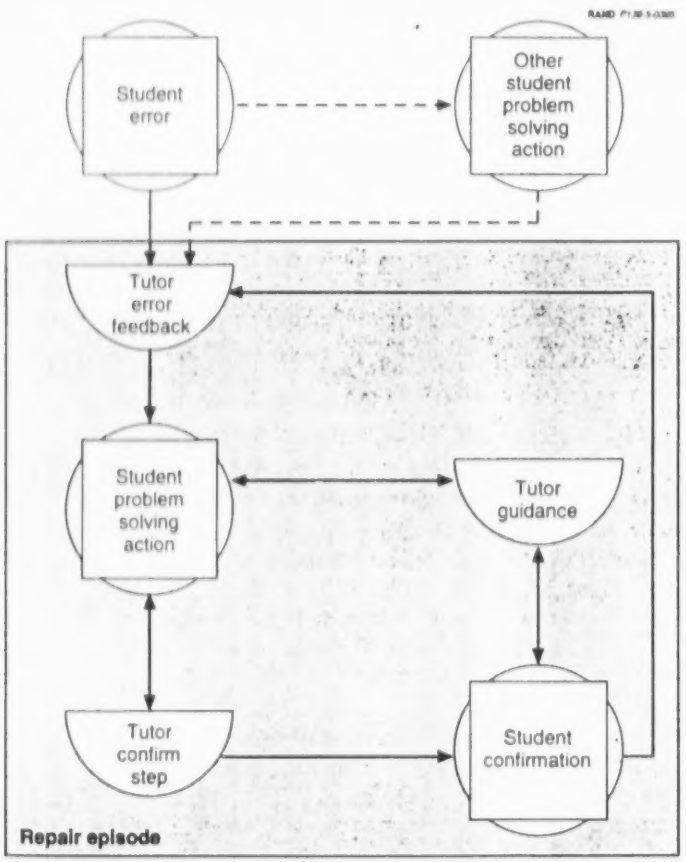

FIGURE 5 The sequence of student and tutor collaborative actions when repairing an error. The dotted lines represent infrequent transitions, occurring less than 100 times in over 1,000 errors represented.

a confirmation, and the tutor offered guidance, perhaps in the form of a new goal to be achieved, which the student attempted to implement with another problem-solving action. This is a clear collaborative enterprise, with the student actively working to overcome the impasse and the tutor offering guidance and confirmations (see Example 13).

Example 13 (collaborative repair of a student error):

1. SPSA	*Student:*	[typing] *(defun classify (var)* *(cond ((null var) 'nil)* *((numberp var) 'number)*
2. TC	*Tutor:*	Right, um hum, just the word number.
3. SPSA	*Student:*	[continues typing] *((t* [error: there should be only one parenthesis.]
4. PBF	*Tutor:*	Now here you have the choice. You could either use *t*=
	Student:	=*t*
	Tutor:	or the predicate=
	Student:	OK
	Tutor:	or list.
5. SPSA	*Student:*	So I don't need that extra.
6. TCS	*Tutor:*	Right, you got it, you got it.
7. TID	*Tutor:*	That's kinda tough 'cause it's a weird pattern that these *conds* have. Great.
8. SPSA	*Student:*	[typing] <DEL><DEL>*t*

There is another path through the error recovery process illustrated in Figure 5. This one, reminiscent of the emphasis of Moore and Swartout (1989) on student difficulties in understanding tutorial feedback, consists of an error, subsequent feedback, a Student Problem-Solving Action, some tutorial guidance, a student confirmation, and then additional error feedback. The interesting part of this path is that error feedback follows a student confirmation. This has to do with the nature of our coding scheme. For example, in one case, the tutor said, "You need the *null* case first," which would be an example of Tutor Error Feedback. If the student did not understand this, the confirmation would be slightly delayed relative to the comment, such as a 1-sec pause before saying, "Umm, OK." People in conversation are very aware of even very short delays (cf. Fox, 1991), so a tutor might interpret this delay as indicative of student confusion, and thus offer more feedback to enable the student to repair the error, such as by saying, "Since *nil* is also a list, you need to test for *null* before using the *listp test*."

These analyses have shown that students do contribute to error recovery, even though tutors often initiate the process. Students and tutors together develop and

implement a solution plan, and each may offer suggestions while the plan is being implemented.

Generality of the Results

In the methodology of this study, we have chosen to emphasize the depth and length of interaction between relatively few tutors and several students rather than a more shallow analysis of more student–tutor pairs. This depth of interaction has allowed us to observe tutorial responses to a wide variety of situations.

Yet, these interactions have taken place in one particular tutorial setting. These students were very bright and highly motivated to learn the material and had not previously experienced difficulty with the domain, as would be the case in a remedial tutoring session. The domain involves acquiring proficiency by solving many exercises, as in other mathematical and scientific domains, although the cognitive complexities of these domains vary. In the programming domain, difficulties often arise as students attempt to express their intentions in a formal notation (Merrill & Reiser, 1994). Tutorial behavior is almost certainly affected by a variety of dimensions, including the features of the student and the domain, and thus the generality of any individual study must be examined carefully. We argue, however, that our results capture general aspects of tutorial behavior applicable to a range of domains wider than computer programming alone. We argue this in two ways: (a) by showing how similar the two tutors were to each other and (b) by pointing out similarities between our tutors and those described in the tutoring literature.

First, the two tutors were very similar to each other. For example, recall that we argued that tutors frequently responded to a correct student action with a Tutor Confirm Step or Tutor Guidance. The tutorial response to a correct student action is one indicator of tutorial style, because different styles lead tutors to respond to student actions differentially. If these tutors display differing styles, then the number of cases where the tutors offered Tutor Guidance versus those where Tutor Confirm Steps were offered will vary. For the female tutor, the number of Tutor Guidance events was equal to 14% of the Tutor Confirm Steps. For the male tutor, the number of Tutor Guidance events was equal to 13% of the Tutor Confirm Steps. This suggests that the tutors are behaving similarly. One can calculate a more general statistic to capture the degree to which the tutors produced similar responses to similar situations by comparing the interaction tables (Castellan, 1979). This technique essentially counts the situations in which each tutor gave the same category of response to the same student utterance type and then adjusts that value for the agreement that would be due to chance, producing a chi-square statistic to test for homogeneity of the tutor–student interactions. Our two tutors behaved in very similar manners toward the students, $\chi^2 < 1$, n.s. No pedagogical strategies were presented to the tutors, so this similarity suggests that experienced tutors may share certain behaviors when

examined over periods of tutoring substantial enough to allow a range of problem-solving situations to occur.

In addition to being similar to one another, our tutors also engaged at one time or another in most of the behaviors previous theorists have highlighted. For example, our tutors offered confirmations, as Fox's (1991) arguments would predict, and offered directive error feedback like the tutors described by Littman et al. (1990) and Schoenfeld et al. (1992). Furthermore, the students and tutors worked together to repair errors, as Fox (1991) suggested, and this collaboration presumably enabled students to feel as though they had mastered the problem-solving difficulties, as Lepper et al. (1990) argued. Thus, our tutors were similar to one another and exhibited at various times the behaviors found in other tutorial studies in other domains. Although characteristics of the students' abilities and the particular character of the domain may certainly influence the frequency and implementation strategies for these tutorial behaviors, the similarity in our tutors and the presence of the range of tutorial behaviors suggest that we have uncovered some general factors tying tutorial behaviors to particular problem-solving situations and that these factors may apply to expert tutoring behaviors in a range of problem-solving domains. In the next section, we present a theory of tutorial guidance that attempts to describe why the tutorial behavior we found should lead to pedagogical advantages.

A THEORY OF TUTORIAL GUIDANCE

Tutors help keep problem solving productive and maximize the learning outcomes of their students by encouraging and supporting successful problem solving and by providing feedback to help students recover from errors. In this section, we review the main findings of this study and present a model of tutorial guidance that accounts for these results.

During problem solving, students encounter impasses and make errors. Sometimes, however, these errors are not visible until many moves after the error occurred. For example, opening a chess game by moving the rook's pawn forward one space may seem reasonable to a novice chess player, and the consequences of this poor choice will not be apparent for some time. How is a problem solver to understand which one of many moves was responsible for the poor outcome? Without this knowledge, the problem solver cannot avoid the error in the next game. This difficulty is often called the *credit assignment* problem. The credit assignment problem occurs when a success or failure arises after several steps; a learner has to figure out which step led to the outcome experienced. This may be a simple task if there are a small number of steps; however, most classes of problems have more than a few steps intervening between an event and the associated marker of success or failure, yielding a large search space.

Clearly, credit assignment could be facilitated by minimizing the number of steps between signals of success or failure and by focusing on the features

potentially responsible. So, for example, a student's learning could be helped if the student could easily determine, after each step, whether it was correct or not.

Anderson and his colleagues (Anderson, Boyle, Corbett, & Lewis, 1990; Anderson et al., 1985) have argued for immediate feedback as an effective pedagogical technique in problem-solving domains due to the difficulties that error recovery can pose for learning (Anderson, Conrad, & Corbett, 1989). Anderson and Corbett (1993) argued that immediate feedback may not yield superior pedagogical outcomes but does lead to more efficient learning. That is, students receiving immediate feedback and those receiving no feedback can learn a domain equally well, but those learning with feedback can master the domain up to 3 times faster.

In fact, our tutors engaged in behaviors that minimized credit assignment to a few events via Tutor Confirm Steps and Tutor Error Feedback. Our tutors responded to both correct and incorrect steps very rapidly, sometimes even on the next event, thus minimizing the space of possible actions that could have led to an error or success. The Tutor Guidance further helped encourage students to continue on promising solution paths and helped focus their search when necessary. Furthermore, the very interactive, efficiently communicated nature of this feedback meant that students could respond to the feedback without unduly interrupting their problem-solving effort. This focusing of students' search behaviors appears to be a central source of pedagogical advantage for individualized instruction.

The tutors responded to different types of errors with different feedback strategies. We next turn to a discussion of the pedagogical benefits of this practice.

There is a trade-off in learning from errors. With each error, there are benefits of self-recovery. Students learn more when they construct explanations for themselves rather than simply encoding a provided explanation (Chi et al., 1989). The costs of floundering—in time, confusion, and frustration—can be serious, however, especially if students do not construct explanations. Indeed, the benefit of self-explanation has been demonstrated primarily among students studying instructional examples rather than among those interspersing self-explanations in their own problem solving. We suggest that tutorial response to errors can be explained by comparing the relative weight of the learning opportunity's potential benefits to its potential costs. We call this relative weighting the *learning consequences* of an error. By examining the learning consequences of each of the three types of errors, we see that the relative weights of costs and benefits predict the type of feedback tutors provided. When there was a great deal of learning possible from self-recovery, tutors allowed students to perform as much of the error recovery as possible. If the costs of repair outweighed the benefits, however, tutors simply told the student how to repair the error, thus keeping the student on track to a solution.

Recall that there were three types of student errors: syntactic, semantic, and goal errors. First, consider syntactic errors. The syntax of computer programming

languages is a source of difficulty for novices, and novices often spend a great deal of time flailing around trying to fix errors arising from mistakes in the notational syntax (Anderson et al., 1985; du Boulay, 1986; Reiser et al., 1991).

Programming language syntax is often a source of great difficulty for novices for a variety of reasons. Among these, language syntax is often arbitrary and has little relation to the ways novices think about real-world analogues of the constructs (Bonar & Soloway, 1985; Trafton & Reiser, 1993b). Furthermore, novices are used to having some margin for error in communication in the real world and sometimes fail to consider exactly how literal one must be when creating a computer program (Bonar & Soloway, 1985; du Boulay, 1986; Pea, 1986). Due to the largely arbitrary nature of syntactic rules, resolving errors typically relies on weak method search or analogy from examples, not from explanation. Thus, students may learn little more by repairing errors themselves than by simply being told how to do the repair. Furthermore, the difficulty of repairing syntactic errors creates a serious danger of floundering. Thus, syntactic errors have low learning consequences. Accordingly, tutors usually just stepped in and offered a Tutor Correction that told the student how to repair the error directly, requiring the student only to implement the repair. This feedback strategy sacrifices the few benefits that might accrue from the repair in favor of keeping the student on a productive repair path, and, because little of the recovery process remained for the student to do, recovering from these errors was quite rapid.

In a semantic error, the student misapplied a LISP function or basic concept. How should tutors support student reasoning after this sort of error? Semantic errors usually received Surface-Feature Feedback that pointed to the incorrect feature evident in the solution. Given how much difficulty novices have locating and repairing errors in programs (Katz & Anderson, 1987–1988; Reiser et al., 1991) and the working memory load caused by the search that interferes with learning (Anderson et al., 1985; Sweller, 1988), one might expect tutors simply to intervene with feedback that performs all the components of the error recovery process in the interest of keeping students from becoming overloaded, as was the case with syntactic errors. However, a critical component of learning a new domain involves acquiring the semantics of operators and reasoning about their interactions when learning to construct more complex plans involving the operators (Merrill & Reiser, 1994; Ohlsson & Rees, 1991; Soloway, 1986). Furthermore, repairing errors often involves reasoning about causes and consequences of the observed erroneous behavior (Katz & Anderson, 1987–1988; Spohrer, Soloway, & Pope, 1985), and successful students generate these sorts of explanations in other problem-solving situations (Chi et al., 1989); so self-generated explanations, with tutorial collaboration if necessary, seem a promising mechanism for students to acquire this knowledge.

Because the tutor only pointed the student to the general location of the error, the student had to recognize what was incorrect in the solution, make an inference about the error's nature, set a goal to repair the error, and then begin to implement

the repair. Thus, students were able to learn about the domain operators through a very focused learning-by-doing session involving recovering from an error. This additional problem-solving effort was reflected in the longer error-recovery episodes. The tutors participated in the entire recovery process as well, serving to keep the student from dangerous floundering (cf. McArthur et al., 1990) and rendering the error-recovery work profitable.

The third class of errors concerned the students' manipulation of the goal structure. Structuring behavior according to goals is a critical feature of learning new problem-solving domains (Anderson, 1983; Newell, 1990; VanLehn, 1990). Indeed, many instructional theorists have argued that helping students maintain and reflect on a goal structure facilitates learning a new domain (Collins & Brown, 1988; Collins, Brown, & Newman, 1989; Koedinger & Anderson, 1990; McArthur et al., 1990; Merrill & Reiser, 1994; Singley, 1990).

Recall that goal errors occurred when the student's manipulation of the goal structure faltered. How did tutors offer the required support? These errors elicited a variety of feedback types. Surface-Feature Feedback, as with semantic errors, helped students to pinpoint the location in the solution where their reasoning had gone awry and to reconstruct a more promising solution plan. In other cases, the support took the form of Tutor Plan-Based Feedback, reminding the student what the current goal should be. This feedback also told the student the location of the error. In a goal error, however, the relevant location was not an explicit component of the solution but rather a subgoal whose implementation was faulty or forgotten. Thus, locating the error required referring to the more abstract goal structure—hence, the appropriateness of Tutor Plan-Based Feedback. Although the tutor could offer additional help throughout the repair, this initial feedback alerted the student to the existence of an error and gave some information about its location, with the tasks of realizing what had been implemented incompletely, setting the goals to perform the repair, and implementing it remaining to be accomplished. Thus, tutors supported students' maintenance of a goal structure in a manner similar to their support of difficulties with operators, by pointing students to a location in a structure that would highlight the error. This allowed students the opportunity to perform much of the error feedback, as in semantic errors, but prevented potential floundering.

Our analyses have shown that tutors modulate their feedback in accordance with learning consequences. Examining the errors that led to our three types of tutorial feedback showed that tutors tended to give explicit corrections that contained directions for the error's repair when an error did not offer the opportunity for significant learning but could lead to unfruitful floundering. Those low-learning-consequences errors offered few benefits of learning by doing, and thus tutors simply told the student how to repair the error, and the students did so. In contrast, when it would be beneficial for the student to explain the erroneous part of a solution and plan the repair, such as in semantic errors, tutors focused students on the error but let them replan a solution for that goal. When the target concept

was more abstract, such as understanding the goal structure, tutors might leave the analysis of the error's feature to the student as well. Because of these strategies, students learned to recover from errors by doing, thus actively exercising, testing, and modifying their problem-solving knowledge, but were also protected from some of the more severe costs of the recovery process by carefully chosen tutorial feedback. This balance of learning by doing and tutorial guidance maximized the effectiveness of students' problem solving and underlay the strong learning gains for tutored students.

In this article, we have presented a model of the relation between student errors and feedback, taking into account the learning consequences of different error types and describing how tutors support ongoing problem solving through confirmations and rapid error flagging. We now return to the views of tutoring advocated by the researchers reviewed earlier.

Fox (1991) argued that confirmatory feedback serves a central role in tutorial discourse. Our results support this, and we argued that the confirmations support problem solving by enabling students to determine more easily which action was responsible for success and which knowledge was faulty. Lepper and his colleagues (Lepper et al., 1990; Lepper & Chabay, 1988) found that tutorial feedback served to help students continue to feel successful, and they argued that this accounted for tutorial pedagogical benefits. In our data, we found a relative lack of feedback that was specifically motivational in character. Presumably, this reflects the differences in ages and domains between the two studies. This reinforces the importance of considering the multidimensionality of tutorial behavior, because changes in some of these dimensions led us to find almost no instances of motivational behaviors that Lepper's tutors found very important. Lepper's tutors were teaching small children who had already had difficulty mastering the arithmetic material. This social situation seems destined to bring out the supportive side of any compassionate instructor. Our students were bright, confident, college-age students learning a domain for the first time, so they presumably required less motivational support. We suggest that our tutors did achieve positive motivational benefits but did so by minimizing student frustrations and helping to keep problem solving productive (cf. Reiser et al., in press) rather than by directly reinforcing students' feeling of success or by helping them explain away difficulties encountered.

Our analyses of tutorial guidance have suggested that part of tutorial effectiveness relies on tailoring feedback to particular problem-solving situations. Indeed, the ability to individualize guidance to particular contexts has been proposed as a central advantage of individualized instruction (B. S. Bloom, 1984; P. A. Cohen et al., 1982; McArthur et al., 1990) and has motivated much work in computerized intelligent tutoring systems (e.g., Anderson et al., 1985; Carbonell, 1970; Goldstein, 1982). In contrast, several studies of tutoring have suggested that tutors follow a fairly uniform simple sequence of pedagogical actions while tutoring students, essentially following straightforward curriculum scripts,

in which they present and query topics, backing up and spending more time as needed to cover troublesome ones (Graesser, 1993; Putnam, 1987; Sleeman, Kelley, Martinak, Ward, & Moore, 1989). In this view of tutoring, curriculum goals that do not vary across students account for most tutorial actions.

Our results suggest, at least for tutoring sessions focusing on problem solving, that this curriculum-script model is an incomplete description of student–tutor interactions. We found that student–tutor dialogues were centered much more around student-initiated events, as they attempted to actively understand new instructional material and solve problems, than around tutorial presentation of material and subsequent querying of student understanding. These more complex patterns of student–tutor interactions are better described by a mixed-initiative dialogue (Carbonell, 1970; Collins et al., 1975) than by the dialogue frames suggested by Graesser (1993) or the curriculum scripts suggested by Putnam (1987). The tutors responded with guidance to support and focus students' reasoning rather than presenting material themselves and then querying students' understanding.

A central issue in this type of tutorial guidance concerns how tutors responded to students' errors. As Putnam (1987) and Lepper and Chabay (1988) pointed out, effective tutors do not appear to attempt to diagnose the faulty reasoning that caused the students' errors. For example, tutors do not propose new problems for students to solve purely for the purpose of discriminating between possible misconceptions that might have caused the error. Tutors also rarely relate possible lines of student reasoning that could have led to the error. Indeed, Sleeman and his colleagues have argued that pointing out a student's misconception to the student is no more effective than simply reteaching the erroneous procedure (Sleeman et al., 1989; Sleeman, Ward, Kelly, Martinak, & Moore, 1991), which might suggest that diagnosis is unnecessary. In these views, effective tutorial response to errors consists of simply reteaching the correct procedure rather than focusing on explanations of why errors occurred.

Our microanalyses of the student–tutorial interactions in problem-solving situations suggest that tutors do more than simply reteach a correct procedure component when students encounter impasses or errors. Our tutors focused on guiding the error repair process rather than on communicating their guesses about the student's misconception. As Merrill et al. (1992) suggested, whether tutors verbalize diagnoses is a less central question than whether (and how) they track student reasoning and determine what type of guidance to provide. The results of our present study demonstrate the ways in which tutors focus students on detecting and repairing errors. Tutors do not simply correct students and review relevant curriculum material. Instead, they collaboratively help the students to understand and repair errors. Furthermore, tutors tailor the timing and specific content of their feedback to the learning consequences of the particular error. Our results demonstrate that tutors intervene less quickly and leave more of the error repair to students when more can be learned from the error repair. Thus,

rather than a path through a curriculum script or dialogue frame, we see very careful tracking of student reasoning and modulation of the timing and nature of feedback, depending on the type of error encountered. We should note, however, that, consistent with the reteaching view, the principal goal seems to be getting the problem solving back on track. These error episodes in general were very short (typically 2 or 3 events) and were always focused on repairing the error rather than exploring it.

The careful adaptation of feedback timing and content suggests that a more complex model than curriculum scripts is needed to explain tutorial behavior. We suggest that tutorial behavior is better modeled by microplans (McArthur et al., 1990; Schoenfeld et al., 1992) in which particular tutorial plans are triggered in response to particular student problem-solving situations. Our analyses suggest that the central microplans for modeling tutorial behavior must track student reasoning and determine when to provide confirmations or additional guidance on correct paths and also specify when to offer feedback to student errors and how much of the error recovery process to perform after this intervention.

CONCLUSION

In this study, we used extensive analysis of many hours of student–tutor discourse to attempt to determine the strategies experienced human tutors use that result in the pedagogical advantages of human tutoring. These analyses suggest that tutors assist students' active problem solving with careful guidance, in which the tutor keeps the student's problem solving on track by providing ongoing confirmatory feedback and new goals to achieve after correct steps and error feedback after errors. Students caught some of their own errors, but if the student did not notice that an error had occurred, the tutor drew the student's attention to it relatively quickly.

We argued that the relative weights of the benefits of self-repair versus the costs of floundering dictated the feedback used by tutors. In situations where an error could lead to floundering and did not offer significant potential for learning, tutors often told the student how to repair the error, thereby leaving only the implementation of the repair to the student. In contrast, tutors offered less support for errors that offered significant benefits of self-repair. Thus, as learning consequences increased, the tutors allowed the students to perform more and more of the error recovery, including constructing their own explanations for the errors and acting on their analyses (cf. Chi et al., 1989). This active self-explanation and problem solving leads students to develop better models of the behavior of operators in the domain.

Tutorial guidance allows an extremely effective style of learning by doing, namely, guided learning by doing. Students can pursue the benefits of actively constructing understandings and solution plans and implementing them with care-

fully modulated guidance from the tutor. Tutors modulate their guidance in response to students' actions and current problem-solving context. Tutors encourage students to continue on profitable paths and warn students of errors through explicit and rapid comments that focus students' attention on sources of errors. When obstacles are encountered, students and tutors collaboratively effect a repair. During this repair, tutors offer guidance and feedback but do so in a manner that encourages students to analyze the error and actively contribute to the repair. This carefully modulated guidance allows the best pedagogical advantages of learning by doing while minimizing the consequent potential costs of self-directed search for a correct answer through a very large problem space. This careful tutorial guidance offered during successful problem solving as well as during difficulties leads tutored students to achieve the substantial cognitive and motivational advantages observed in individualized tutoring.

ACKNOWLEDGMENTS

This work was supported in part by contracts MDA903–87–K–0652 and MDA903–90–C–0123 to Princeton University and contract MDA903–92–C–0114 to Northwestern University from the Army Research Institute and by a grant from the Spencer Foundation. The views and conclusions in this article are those of the authors and should not be interpreted as representing the official policies of those institutions.

The comments of Peter Pirolli and an anonymous reviewer significantly improved this article. We also gratefully acknowledge the programming assistance of Jeremiah Faries, comments from David McArthur, Michael Ranney, and Richard Beckwith, and assistance from Holly Hillman and Jason Thompson. Diane Schwartz was of invaluable assistance with several of the figures presented in this article.

Portions of these analyses were presented at the annual meeting of the American Educational Research Association, March 1992.

REFERENCES

Anderson, J. R. (1983). *The architecture of cognition.* Cambridge, MA: Harvard University Press.

Anderson, J. R. (1987). Skill acquisition: Compilation of weak-method problem solutions. *Psychological Review, 94,* 192–210.

Anderson, J. R. (1989). The analogical origins of errors in problem solving. In D. Klahr & K. Kotovsky (Eds.), *Complex information processing: The impact of Herbert A. Simon* (pp. 343–372). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Anderson, J. R., Boyle, C. F., Corbett, A. T., & Lewis, M. W. (1990). Cognitive modeling and intelligent tutoring. *Artificial Intelligence, 42,* 7–49.

Anderson, J. R., Boyle, C. F., & Reiser, B. J. (1985). Intelligent tutoring systems. *Science, 228,* 456–562.

BEST COPY AVAILABLE

Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1989). Skill acquisition and the LISP tutor. *Cognitive Science, 13*, 467–505.

Anderson, J. R., & Corbett, A. T. (1993). Tutoring of cognitive skill. In J. R. Anderson (Ed.), *Rules of the mind* (pp. 235–255). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Anderson, J. R., Corbett, A. T., & Reiser, B. J. (1987). *Essential LISP*. Reading, MA: Addison-Wesley.

Anderson, J. R., & Jeffries, R. (1985). Novice LISP errors: Undetected losses of information from working memory. *Human–Computer Interaction, 1*, 107–131.

Bakeman, R., & Gottman, J. M. (1986). *Observing interaction: An introduction to sequential analysis*. Cambridge, England: Cambridge University Press.

Bloom, B. S. (1984). The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher, 13*, 4–16.

Bloom, L., Rocissano, L., & Hood, L. (1976). Adult–child discourse: Developmental interaction between information processing and linguistic knowledge. *Cognitive Psychology, 8*, 521–551.

Bonar, J. G., & Soloway, E. (1985). Preprogramming knowledge: A major source of misconceptions in novice programmers. *Human–Computer Interaction, 1*, 133–161.

Carbonell, J. R. (1970). AI in CAI: An artificial intelligence approach to computer-aided instruction. *IEEE Transactions on Man–Machine Systems, 11*, 190–202.

Castellan, N. J., Jr. (1979). The analysis of behavior sequences. In R. B. Cairns (Ed.), *The analysis of social interactions: Methods, issues, and illustrations* (pp. 81–116). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Chi, M. T. H., Bassok, M., Lewis, M. W., Reimann, P., & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science, 13*, 145–182.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement, 20*, 37–46.

Cohen, P. A., Kulik, J. A., & Kulik, C. C. (1982). Educational outcomes of tutoring: A meta-analysis of findings. *American Educational Research Journal, 19*, 237–248.

Collins, A., & Brown, J. S. (1988). The computer as a tool for learning through reflection. In H. Mandl & A. Lesgold (Eds.), *Learning issues for intelligent tutoring systems* (pp. 1–18). New York: Springer-Verlag.

Collins, A., Brown, J. S., & Newman, S. E. (1989). Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. In L. B. Resnick (Ed.), *Knowing, learning, and instruction: Essays in honor of Robert Glaser* (pp. 453–494). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Collins, A., & Stevens, A. L. (1982). Goals and strategies of inquiry teachers. In R. Glaser (Ed.), *Advances in instructional psychology* (Vol. 2, pp. 65–119). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Collins, A., Warnock, E. H., & Passafiume, J. J. (1975). Analysis and synthesis of tutorial dialogues. In G. H. Bower (Ed.), *The psychology of learning and motivation* (pp. 49–87). New York: Academic.

du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research, 2*, 57–73.

Ericsson, K. A., & Simon, H. A. (1984). *Protocol analysis: Verbal reports as data*. Cambridge, MA: MIT Press.

Faries, J. M. (1991). Reasoning-based retrieval of analogies (Doctoral dissertation, Princeton University, 1991). *Dissertation Abstracts International, 52*, 2329B.

Fisher, C. (1991). Protocol analyst's workbench: Design and evaluation of computer-aided protocol analysis (Doctoral dissertation, Carnegie Mellon University, 1991). *Dissertation Abstracts International, 52*, 3277B.

Fox, B. A. (1991). Cognitive and interactional aspects of correction in tutoring. In P. Goodyear (Ed.), *Teaching knowledge and intelligent tutoring* (pp. 149–172). Norwood, NJ: Ablex.

Gentner, D. (1983). Structure mapping: A theoretical framework for analogy. *Cognitive Science, 7*, 155–170.

Gick, M. L., & Holyoak, K. J. (1980). Analogical problem solving. *Cognitive Psychology, 12*, 306–355.

Goldstein, I. P. (1982). The genetic graph: A representation for the evolution of procedural knowledge. In D. H. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems* (pp. 51–77). London: Academic.

Graesser, A. (1992). *Questioning mechanisms during complex learning*. Memphis, TN: Memphis State University.

Graesser, A. C. (1993, August). *Dialogue patterns and feedback mechanisms during naturalistic tutoring*. Paper presented at the annual meeting of the Cognitive Science Society, Boulder, CO.

Graesser, A. C., Person, N. K., & Huber, J. D. (1993). Question asking during tutoring and in the design of educational software. In M. Rabinowitz (Ed.), *Cognition, instruction, and educational assessment* (pp. 149–172). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Grice, H. P. (1975). Logic and conversation. In P. Cole & J. L. Morgan (Eds.), *Syntax and semantics: Vol. 3. Speech acts* (pp. 41–58). New York: Academic.

Katz, I. R., & Anderson, J. R. (1987–1988). Debugging: An analysis of bug location strategies. *Human–Computer Interaction, 3*, 351–399.

Kerry, T. (1987). Classroom questions in England. *Questioning Exchange, 1*, 32–33.

Koedinger, K. R., & Anderson, J. R. (1990). Abstract planni: g and perceptual chunks: Elements of expertise in geometry. *Cognitive Science, 14*, 511–550.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). *Universal subgoaling and chunking: The automatic generation and learning of goal hierarchies*. Hingham, MA: Kluwer Academic.

Lepper, M. R., Aspinwall, L., Mumme, D., & Chabay, R. W. (1990). Self-perception and social perception processes in tutoring: Subtle social control strategies of expert tutors. In J. M. Olson & M. P. Zanna (Eds.), *Self-inference processes: The Sixth Ontario Symposium in Social Psychology* (pp. 217–237). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Lepper, M. R., & Chabay, R. W. (1988). Socializing the intelligent tutor: Bringing empathy to computer tutors. In H. Mandl & A. Lesgold (Eds.), *Learning issues for intelligent tutoring systems* (pp. 242–257). New York: Springer-Verlag.

Lewis, M. W., & Anderson, J. R. (1985). Discrimination of operator schemata in problem solving: Learning from examples. *Cognitive Psychology, 17*, 26–65.

Littman, D. (1991). Tutorial planning schemas. In P. Goodyear (Ed.), *Teaching knowledge and intelligent tutoring* (pp. 107–122). Norwood, NJ: Ablex.

Littman, D., Pinto, J., & Soloway, E. (1990). The knowledge required for tutorial planning: An empirical analysis. *Interactive Learning Environments, 1*, 124–151.

Mayer, R. E., Dyck, J. L., & Vilberg, W. (1986). Learning to program and learning to think: What's the connection? *Communications of the ACM, 29*, 605–610.

McArthur, D., Stasz, C., & Zmuidzinas, M. (1990). Tutoring techniques in algebra. *Cognition and Instruction, 7*, 197–244.

McKendree, J. (1990). Effective feedback content for tutoring complex skills. *Human–Computer Interaction, 5*, 381–413.

Merrill, D. C., & Reiser, B. J. (1994). *Reasoning-congruent learning environments: Scaffolding learning by doing in new domains*. Manuscript submitted for publication.

Merrill, D. C., Reiser, B. J., Ranney, M., & Trafton, J. G. (1992). Effective tutoring techniques: A comparison of human tutors and intelligent tutoring systems. *Journal of the Learning Sciences, 2*, 277–306.

Moore, J. D., & Swartout, W. R. (1989, August). *A reactive approach to explanation*. Paper presented at the 11th International Joint Conference on Artificial Intelligence, San Mateo, CA.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.

Ohlsson, S., & Rees, E. (1991). The function of conceptual understanding in the learning of arithmetic procedures. *Cognition and Instruction, 8*, 103–179.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books.

Pea, R. D. (1986). Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research, 2*, 25–36.

Pirolli, P. (1991). Effects of examples and their explanations in a lesson on recursion: A production system analysis. *Cognition and Instruction, 8,* 207–259.

Putnam, R. T. (1987). Structuring and adjusting content for students: A study of live and simulated tutoring of addition. *American Educational Research Journal, 24,* 13–48.

Reiser, B. J., Beekelaar, R., Tyle, A., & Merrill, D. C. (1991, August). *GIL: Scaffolding learning to program with reasoning-congruent representations.* Paper presented at the International Conference of the Learning Sciences, Evanston, IL.

Reiser, B. J., Copen, W. A., Ranney, M., Hamid, A., & Kimberg, D. Y. (in press). Cognitive and motivational consequences of tutoring and discovery learning. *Cognition and Instruction.*

Reiser, B. J., Kimberg, D. Y., Lovett, M. C., & Ranney, M. (1992). Knowledge representation and explanation in GIL, an intelligent tutor for programming. In J. H. Larkin & R. W. Chabay (Eds.), *Computer-assisted instruction and intelligent tutoring systems: Shared goals and complementary approaches* (pp. 111–149). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Scardamalia, M., Bereiter, C., McLean, R. S., Swallow, J., & Woodruff, E. (1989). Computer-supported intentional learning environments. *Journal of Educational Computing Research, 5,* 51–68.

Schank, R. C., & Leake, D. B. (1989). Creativity and learning in a case-based explainer. *Artificial Intelligence, 40,* 353–385.

Schoenfeld, A. H., Gamoran, M., Kessel, C., & Leonard, M. (1992, April). *Toward a comprehensive model of human tutoring in complex subject matter domains.* Paper presented at the meeting of the American Educational Research Association, San Francisco.

Singley, M. K. (1990). The reification of goal structures in a calculus tutor: Effects on problem solving performance. *Interactive Learning Environments, 1,* 102–123.

Sleeman, D., Kelley, A. E., Martinak, R., Ward, R. D., & Moore, R. L. (1989). Studies of diagnosis and remediation with high school algebra students. *Cognitive Science, 13,* 551–568.

Sleeman, D., Ward, R. D., Kelley, E., Martinak, R., & Moore, J. (1991). Overview of recent studies with PIXIE. In P. Goodyear (Ed.), *Teaching knowledge and intelligent tutoring* (pp. 173–185). Norwood, NJ: Ablex.

Soloway, E. (1986). Learning to program = learning to construct mechanisms and explanations. *Communications of the ACM, 29,* 850–858.

Spohrer, J. C., Soloway, E., & Pope, E. (1985). A Goal/Plan analysis of buggy PASCAL programs. *Human–Computer Interaction, 1,* 163–207.

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science, 12,* 257–285.

Trafton, J. G., & Reiser, B. J. (1993a, August). *The contributions of studying examples and solving problems to skill acquisition.* Paper presented at the annual meeting of the Cognitive Science Society, Boulder, CO.

Trafton, J. G., & Reiser, B. J. (1993b). *Novices' use of forward and backward reasoning in a new problem solving domain.* Unpublished manuscript, Northwestern University, Evanston, IL.

VanLehn, K. (1988). Toward a theory of impasse-driven learning. In H. Mandl & A. Lesgold (Eds.), *Learning issues for intelligent tutoring systems* (pp. 19–41). New York: Springer-Verlag.

VanLehn, K. (1990). *Mind bugs: The origins of procedural misconceptions.* Cambridge, MA: MIT Press.

VanLehn, K., Jones, R., & Chi, M. T. H. (1992). A model of the self-explanation effect. *Journal of the Learning Sciences, 2,* 1–59.

BEST COPY AVAILABLE

APPENDIX A: INSTRUCTIONS FOR CODERS AND RELIABILITY CODERS

Segmentation Instructions

1. Break a new segment if and only if you can clearly demonstrate a need to do so. In other words, a segment should be continued until a concept is introduced that was not present before.
2. Pronouns are tricky. If you are segmenting and run across a new usage of some pronoun (e.g., "it"), try to find the meaning of the pronoun in the current segment. Only break a segment when a pronoun is introduced if the pronoun cannot be bound to a noun in the current segment.
3. Segments *can* continue across interruptions if the interruption is not heeded by the speaker. If something is said as an interruption (no matter how small) that the speaker responds to *in any way*, a new segment must be made for the interruption.
4. Segmentation based solely upon typing must be done very carefully, because there is very little information in a typing episode. Thus, a segment can be made if the student completes a complete LISP action (*defun*, function call) or if there is conversation during the typing, but never within a *defun*.
5. Segmentation is independent of categorization. Don't worry about what a segment will be—break according to the rules above.

Coding Instructions

1. Each segment must receive 1 and only 1 of the codes that appear in the attached pages.
2. Categorize based upon *what was said*, not what you think the utterance meant. The codes are grouped into questions and assertions. If you are categorizing a question, the code you apply must be one of the question codes, etc.
3. The default categories are marked. Unless you have definite reason to do otherwise, you should use the default category.
4. Do not "read into" category definitions. If an utterance does not quite fit category A, it does not fit! Do not stretch the categories to fit an utterance.
5. Sometimes a typing episode will be in the middle of a segment. Encode this segment according to the most important element. For example, if the person is simply saying what they are typing, the segment should be categorized as typing, and so forth.

APPENDIX B: DEFINITIONS OF CODING CATEGORIES

The categories are grouped here as they were discussed in the section on Discourse Analysis Methods. All examples are taken verbatim from the actual protocols.

The Student Constructs a Solution (Student Problem-Solving Action)

Student Correction. This category consists of self-corrections by the student, in which the student has made an error, but immediately recognizes it and suggests how to fix it.

Oh, I need a quote before that!

Whoops, I need to add a parenthesis there.

Student Elaboration. An utterance is categorized as an Elaboration if the student is answering a question, without providing actual data (which would be a Student Simulate Process, described later), or is simply adding to the information already presented in the conversation. Thus, Student Elaboration is a default category for student-to-tutor assertions that are task related, but do not fit any other category.

OK, an empty list.

The rest of the sentence after by . . .

Student Example. The student produces a concrete example to demonstrate some point or to ask a question.

What about "*a*" and "*(b c d)*"?

What about *nil* then?

Student Focuses Attention. The student causes some item in the book or on the screen to become the focus of the conversation.

And this *cond* is the thing we're finding.

Oh, they're talking about this!

Student Indicates Difficulty. The student remarks that a problem is difficult (or long, etc.) or makes some comment indicating that he or she thinks the next section will be hard.

Writing this would be a problem.

Boy this is long!

Student Indicates Lack of Understanding. These utterances tell the tutor that the student is confused. This could be done directly, or indirectly, such as through several repetitions of the same word, without making any progress toward answering the question (as in the second example).

I don't understand what they mean here.

Oh, parameters, parameters are . . . umm . . . they're uhh . . .

Student Reads. The student reads from the textbook. This is marked in the transcripts by markers such as "[2.1]." The numbers reflect the section of the text being read.

Student Refers. The student refers to other material to shed light on the current situation. The material could be a previous problem, a section of the text that was already read. The important aspect is that the student uses previous work to cast light on the current situation.

This is just like what we did yesterday, the *pal* thing.

Actually, this would have been true if this is greater than. It's just the same thing.

Student Sets Goal. The student sets a goal for what to do next or indicates how to do the next step. Thus, this category includes both statements about goals and the plans that can achieve them.

So they want us to write down for each//

Now I have to put the quote.

Student Types. The student types into the LISP interpreter or writes on the paper. This is denoted either by "(writing)" or by the time the student began typing, such as "[20:26:23]."

[20:10:45]

Then this, (writing)

The Student Asks for Help From the Tutor

Assist Plan Assertion. This category contains utterances that request an evaluation of the student's plan or understanding of the problem.

But we can't do, can't we do *cons* twice or something?

Do I . . . do I go *zerop num*?

Assist Plan Question. This category consists of utterances that are requests for the tutor's help in deciding what to do next. These utterances can be implied or actual questions.

Now I should do . . .

I don't know what to do now.

Assist Understanding. This category contains utterances that ask for the tutor to evaluate the student's understanding of a LISP concept. These utterances can be questions or implied questions.

Do you mean to say it can't be more than one, right?

But what do you mean by variable, this is the variable?

Student Informational Request. This is the most conservative student-to-tutor request category. If there is no reason to think that a student request is either an Assist Understanding or an Assist Plan, then the utterance should be coded as this. This category also includes requests such as how to use the editor, how to type parentheses.

Exit-with-save was what, F9?

But why are they saying that it's surprising?

The Student Indicates That the Tutor's Utterances Were Understood

Student Confirmation. The student says something to indicate that he or she is following along with what the tutor said, either by some sort of restatement or a simple "OK."

It can be very complex, I understand.

Yes, this is what we want.

The Student Checks the Current Answer

Student Simulates Process. This category is similar to Elaborate, with a crucial difference. This one contains utterances that require the student to work through the behavior that the computer would execute on the current example, either verbally or nonverbally. In other words, utterances that describe how LISP would actually process some definition are classified here. In addition, utterances that produce data output from functions fall here as well, because to produce data, the student must have "run the function" in his or her head.

And then after that's done, see I want to put *d* again, and again the same thing.

Oh this will give me *(plum apple cantaloupe grape)*.

Miscellaneous Non-Task-Related Student Utterances

Student Comments. This category contains unclassifiable utterances and unrelated talk. In addition, if a student makes an assertion that cannot be put into another category because it is unclear what is being said, the utterance is categorized here.

Oh, hi there, just a second.

Well, you, this is//

The Tutor Performs a Portion of the Problem Solving

Tutor Example. This is the tutorial version of Student Example; thus, this category consists of the tutor proposing a concrete example to be worked on. This is listed as a question, because these utterances often take the form of "What about list *a b c*?"

OK, how about the input *nil* there?

What about using *"(a b c d)"* there?

Tutor Focuses Attention. This is the tutorial version of Student Focuses Attention and consists of the tutor making something the topic of conversation, without giving any new information about it. Thus, the tutor could be pointing to something in the text or to something that had just been said.

This is remember, a variable now.

And *defun* always returns the name of the function.

Tutor Reads. This is the tutorial version of Student Reads. The tutor reads from the textbook, and the section read is denoted by markers such as "[2.1]."

Tutor Refers. This is the tutorial version of Student Refers. These utterances involve the tutor bringing previous work to bear on the current situation. The work could be a previous part of the textbook, a previous problem, or even an alternate way of solving a problem.

Just like *car* and *cdr* and *cons* have names, if you're defining a new function you need to give it a name.

It's just like what you were doing yesterday, figuring out what you want to do.

Tutor Types. This is the tutorial version of Student Types, and is marked either with the time the typing occurred or with "(writing)."

[20:10:45]

Then this, (writing)

The Tutor Offers Guidance for the Student's Ongoing Problem Solving

Tutor Confidence Builder. The tutor expresses confidence in the student's ability to solve the problem or offers praise to the student about a specific problem-solving success.

It's really good that you thought to put this first!

Yeah, but the last one will be easy for you.

Tutor Hint. This category captures tutorial utterances that hint at the next step but do not give it fully. Thus, these utterances are similar to Tutor Sets Goal (defined later) but are not as directive—rather, they just suggest a course of action to be considered.

Remember, you have the *less-than* and *greater-than* predicates.

For example, what about the predicate that tests *atom*?

Tutor Indicates Difficulty. This is the tutorial version of Student Indicates Difficulty. This category includes utterances that describe the current problem as hard or long, and so on, or tell the student that the next set of problems will be very difficult.

But you see, this is the complex part.

And I have to warn you, these are getting tougher, so don't worry if it takes you longer.

Tutor Sets Goal. This is the tutorial version of Student Sets Goal and includes assertions about what to do next or how to do it. The statement may refer to a new goal or to a plan that achieves a stated goal.

So now, if you, you could type this in. We can do some examples with it.

So you'll need the function name, then the parameters, and then the body.

Tutor Supportive Statement. This category contains utterances that are designed to make the student aware that the tutor is there to help if needed.

And then, whenever you have a question, just let me know.

I can help you if you need it.

The Tutor Confirms a Student Step

Tutor Confirmation. This is the tutorial version of Student Confirmation. Thus, these utterances indicate that the tutor is following along with what the student is saying or doing. This could be done via a restatement or a simple "OK." Notice that this category includes the tutor telling the student that the step is right or saying that the student's last comment was understood.

Right, uh huh.

Yes, this part.

Tutor Elaboration. This is the tutorial version of Student Elaboration. Tutor utterances are categorized as Elaborations if the tutor answers a question, without providing explicit data or output of a function (which would be a Tutor Simulates Process, described later), or is simply saying something that adds to the information present in the discussion. This is a default tutorial utterance, so if a tutorial assertion seems on task, but does not fit any other category, it should go here.

Right, because it's embedded in this bigger list, but when you take it out, this is just like a list with two separate . . .

And divide is slash which is near the question mark.

The Tutor Gives Error Feedback After an Incorrect Student Step

Tutor Correction. This is the tutorial version of Student Correction and involves the tutor telling the student exactly how to fix an error. The presence of a direct statement of how to fix the error is the marker of a Tutor or Student Correction. The subject of the correction and the amount of explanation in the correction can vary, but there must be an explicit direction for fixing the error.

And one more for this one.

No, in a list.

Tutor Plan-Based Feedback. This is one of the forms of tutorial error feedback. This type of feedback requires that the tutor know what the student was trying to do when the error occurred. If the student makes an error and the tutor responds to that error by referring the student to the goal that the student should have been working on, that utterance should be categorized as a Tutor Plan-Based Feedback. These utterances are similar to Tutor Sets Goal (defined earlier), except that they occur after an error and refer to the goal the student should have been following.

Well, you'll want to use *and* and *or*, not two *or*'s.

Oh wait, you don't want to, you don't want to return now.

Tutor Surface-Feature Feedback. This is another type of tutorial error feedback. This type of feedback points the student to the feature of the solution that is incorrect. The feature could be syntactic or it could be relating to the function the student chose, and so forth. The identifying elements of this category are that an error has occurred and that the tutor simply makes the student aware of the surface feature that is wrong.

Well, actually *listp* would return true for an empty list, also.

Quote why ['*why*] is not a function.

The Tutor Attempts to Assess the Student's Understanding of a Topic

Tutor Probe. The tutor tries to determine what the student knows about some topic. The topic could be a LISP function, a problem, and so forth.

OK, now do you remember that?

OK, so how many elements?

Tutor Prompt. This category is for tutor utterances that are asking for the student's next step. So the tutor could be asking for the next step of a problem, of an example, or of the understanding of a problem.

So what will that part return?

Cons that atom into the list, and then [pause]?

The Tutor Helps the Student Check the Current Answer

Tutor Simulates Process. This is the tutorial version of Student Simulates Process and includes utterances that include the production of data in the manner that LISP would. That is, the tutor works through the behavior the computer would execute on the LISP code; this could be verbal or nonverbal. If the tutor produces actual data output from a function, the code must have been run in the tutor's head, so the utterance is categorized here.

So this is not a list, and it returns *nil*.

If you call the function *atom*, on *nil*, it returns *true*, because it says that *nil* is an atom. It also returns *nil* if we do it on *listp*.

Miscellaneous Non-Task-Related Tutor Utterances

Tutor Comment. This is the tutorial version of Student Comment. This category consists of tutor utterances that were either unrelated to the task or uninterpretable.

Whoops, let me turn this off.

Do you want a drink?

APPENDIX C
Transitions Between Events

	SPSA	*S Asks for Help*	*S Confirms*	*S Checks Answer*	*TPSA*	*T Guidance*	*T Confirms Step*	*T Error Feedback*	*T Assesses Understanding*	*T Checks Answer*
SPSA	623	136	30	75	153	513	1,744	397	156	78
S Asks for Help	34	0	6	3	19	68	501	67	23	23
S Confirms	282	56	33	29	130	311	807	36	92	201
S Checks Answer	40	1	4	28	10	25	249	59	17	33
TPSA	120	2	168	21	15	18	42	17	15	22
T Guidance	561	63	343	30	29	93	112	30	33	30
T Confirms Step	1,495	302	923	168	72	273	493	47	58	118
T Error Feedback	281	43	222	25	14	48	64	14	18	17
T Assesses Understanding	37	26	88	80	4	15	22	5	6	6
T Checks Answer	23	124	230	33	14	7	22	61	9	24
Total	3,506	753	2,047	495	460	1,371	4,056	733	427	552

Note. The numbers in this table are the frequencies of transitions between events. The labels along the left side are the first event, and the labels along the top are the subsequent event. For example, the leftmost number of the second row is 34. This number indicates that there were 34 instances where a Student Problem-Solving Action followed a Student Asks for Help. Event labels in general refer to the higher-order categories of student (S) or tutor (T) actions. SPSA = Student Problem-Solving Action; TPSA = Tutor Problem-Solving Action.

BEST COPY AVAILABLE